Multi-objective Constrained Optimization for Decision Making and Optimization for System Architectures

by

Maokai Lin

Submitted to the School of Engineering
in partial fulfillment of the requirements for the degree of

Master of Science in Computation for Design and Optimization

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

February 2010

Author .....
School of Engineering
December 2nd, 2009

Certified by .....
Edward F. Crawley
Ford Professor of Engineering
Thesis Supervisor

Certified by .....
Brian C. Williams
Professor of Aeronautics and Astronautics
Thesis Supervisor

Accepted by .....
Karen Willcox
Associate Professor of Aeronautics and Astronautics
Codirector, Computation for Design and Optimization Program

Multi-objective Constrained Optimization for Decision Making and Optimization for System Architectures

by

Maokai Lin

Submitted to the School of Engineering
on December 2nd, 2009, in partial fulfillment of the
requirements for the degree of
Master of Science in Computation for Design and Optimization

Abstract

This thesis proposes new methods to solve three problems: 1) how to model and solve decision-making problems, 2) how to translate between a graphical representation of systems and a matrix representation of systems, and 3) how to cluster single and multiple Design Structure Matrices (DSM).

To solve the first problem, the thesis provides an approach to model decision-making problems as multi-objective Constraint Optimization Problems (COP) based on their common structures. A set of new algorithms to find Pareto front of multi-objective COP is developed by generalizing upon the Conflict-directed A* (CDA*) algorithm for single-objective COPs. Two case studies – Apollo mission mode study and earth science decadal survey study – are provided to demonstrate the effectiveness of the modelling approach and the set of algorithms when they are applied to real-world problems.

For the second problem, the thesis first extends classical DSMs to incorporate different relations between components in a system. The Markov property of the extended DSM is then revealed. Furthermore, the thesis introduces the concept of "projection", which maps and condenses a system graph to a DSM based on the Markov property of DSM.

For the last problem, an integer programming model is developed to encode the single DSM clustering problem. The thesis tests the effectiveness of the model by applying it to a part of a real-world jet engine design project. The model is further extended to solve the multiple DSM clustering problems.

Thesis Supervisor: Edward F. Crawley
Title: Ford Professor of Engineering

Thesis Supervisor: Brian C. Williams
Title: Professor of Aeronautics and Astronautics

Acknowledgments

I would like to thank my two thesis supervisors, Professor Edward Crawley and Professor Brian Williams. Professor Crawley's patient guidance and help in the last two years changes me from an undergraduate student who learns passively to a graduate student who starts to carry out independent research. From him, I learn how to think, write and present like a good researcher. I also constantly benefit from discussions with him, from which I get novel ideas all the time. Moreover, as such a caring advisor, he pays special attention to my international background and helps me merge quickly into the completely new academic environment. Professor Williams leads me into the Artificial Intelligence world and introduces me the constraint optimization problems and algorithms. I learn from him how to do research in a spiral manner which keeps me focusing on the main problems without being astray in minor details. He also spent a great amount of time helping me shape my thesis: he even read, commented and mailed me my second draft on his way to the summer vacation. I really appreciate his guidance and encouragement in the whole academic year.

Especially, I would like to thank Gustavo Pinheiro, who has been cooperating with me on the research project since last year. He is a brilliant and enthusiastic researcher. I get so many new ideas from him. I am also grateful for the help from Bill Simmons, who helped me to start my research at MIT smoothly and taught me the fundamentals of system architecture.

I also want to thank Professor Jaime Peraire and Robert Freund, the two former co-directors of the CDO program, for offering many valuable advices on my academic study and research. I would like to thank Andreas Hofmann, who read my thesis and gave me valuable comments. I thank Theo Seher and Derek Rayside for helping me better understand the CSP algorithms and the earth science decadal survey study, and I thank Justin Colson for his help with the research of system representation translation. Kathi Brazil is always very nice to me and offers every help I need. Her office is one of the places that I would love to stay. I also like to stay in the CDO headquarter, where I can talk to Laura Koller, who helped me through all the

academic procedures and resolved all the troubles I made. I also thank all the people in the system architecture group, who are active in research and always willing to help. I am so glad I get to know everyone of them.

My family provided me much support in the last two years, even though they are a hemisphere away. My parents always care about my study and life and console me when I am depressed or anxious. My grandparents also care about my life here. My grandfather writes me many emails asking about my health and study.

Last but not least, I really appreciate the generous support from Draper Laboratory in the last two years. I especially appreciate the help of Nick Harrison and Jana Schwartz, the two coordinators of the research project. They provide resources to help me better understand the goal of the project, and gave me many valuable comments to make the research useful for real-world problems.

Contents

List of Figures

List of Tables

Chapter 1

Overview

1.1 Motivations and Objectives

People make decisions everyday. The decisions could be as simple as choosing which brand of clothes to buy, and could be as intertwined as choosing the mission mode for the Apollo project in 1960s. People mostly view decision making as an art, but actually, in many decision making scenarios, people can use quantitative skills to help make better choices during a part of the decision-making process, if not through all of it. Especially, when people are facing a large-scale and/or complex system and getting lost in numerous possible options, the decision making "science" can effectively help them narrow down the choices to a manageable range and even to a small set of options that are near optimal.

In this section, I first explain why decision making in complex systems is hard, which explains why we need aiding tools for decision making. I then discuss the difficulty of solving large-scale decision-making problems, and the concern of uncertainties when solving the problems. I also discuss two categories of tools for modeling complex systems and the need of building a bridge between them. Finally, I discuss the need of decomposing and reintegrating the complex systems, and the tools that are needed in such a process.

1.1.1 Difficulties in Decision Making

Think of yourself as the chief engineer of designing the mission mode of Apollo. The goal is to land man on the moon and send them back to the earth safely. To achieve this ambitious goal, you need to choose the type of the rocket, whether to send the spaceship into the lunar orbit or directly to the lunar surface, how many crew members should be on the command module and the lunar module, etc.. You also have to consider how to increase the reliability of the mission, how to control the expense within the budget, and many more. A large number of dazzling alternatives would pour into your mind and might easily flood your analysis ability.

Such a decision making process is hard, because so many decisions have to be made, so many constraints have to be considered, and so many goals have to be achieved. Certainly, different problems may involve different difficulties, but generally, there are three basic sources of difficulties[7]:

First, the system itself is very complex. For example, to make a choice of the Apollo mission mode, one has to consider tens of major processes and hundreds of detailed processes. Many of these processes involves a few different options, and choices of the options are bounded by engineering constraints. Similar situations exist in almost all large-scale system design – a large number of decisions to make, many possible options for the decisions, a large number of constraints that limit the possible combinations of the options chosen for the decisions, a set of objectives that had better be optimized when making the choices. Thus, the complexity is one of the most important sources of difficulty in decision making.

Second, the decision making often involves two or more objectives that are conflict with each other to some extent. For example, when considering the mission of Apollo, the reliability and cost are two of the most important things that should be considered. However, the two objectives could not be both optimized at the same time: the improvement of the reliability might drive up the cost significantly, and on the other hand, saving some money might compromise the overall reliability of the mission. As said by Mencius, "One cannot get fish and bear's paw at the same time." How to

balance the gains and losses between different objectives is another source of difficulty in decision making.

Last but not least, decision making is usually affected by uncertainties. Some critical data like the failure rate of a mission mode could be just an estimation, or could change with the the environment on the launching day. Other than the best option, decision makers usually need to examine the alternatives that are not the best, but is good enough and less risky, or in other words, robust enough to sustain a certain degree of fluctuation.

All the issues, of course, could be addressed by human brains, but it will be faster and less error-prone to let computers carry out the works, and leave the precious manpower to refine a relatively small number of results. The need of designing, implementing and improving such decision-aid tools is the main motivation of my work.

1.1.2 Solving Large-Scale Decision Making Problems

As mentioned in the last section, one of the main goal of the decision-aid tools is to help people find the good decisions in a complex system. This is not an easy task for the computer though. In a complex system in the real world, there could be hundreds of decisions to be made, and each of them has several options. A straightforward method of finding the good solutions is to enumerate all of the possible solutions, filter the ones that do not satisfy the constraints, and sort the results based on our objectives. This approach, however, does not work well in practice because of the large amount of computational time and storage space it needs: The total number of possible combinations is so large that it will take the computer a long time to find all the solutions and a large space to store them. For example, if there are twenty main parts in a car, and we have three options for each of the parts, the total number of possible combination of the twenty parts is 3^{20}, which is about 3.5 billion! One can argue that solving such a problem might just take a day, but what if 1) the number of parts is not twenty, but fifty or a hundred? 2) the engineer has to modify the model frequently? The time spent on solving this problem is too long to be afforded by any

manufacturer.

One might have noticed that not all of the enumerated solutions are used. In most of the cases, even if we enumerate billions of solutions, we probably just need the best twenty of them, if no less. This observation sets one of the objectives for designing the decision-aid tools: to develop an algorithm to search and store only the "good" solutions.

Another important functionality that the decision makers need is to identify the trade-offs between different objectives, especially when the objectives have conflicts with each other. The "good" results are the options that are not worse than some other option in every aspect. For example, when buying a computer, people will compare the price and performance. There could be hundreds of different computers in the market, but people would not be interested in those that are expensive but running slow. Or in other words, the options whose objective values are all worse than another option are not what we need. The decision makers need the tool that could help them filter the options and leave only the "good" ones.

In addition, a decision process is often divided into two stages: the first is to focus on the factors that are critical to the system and temporarily ignore the less important details. When the key decisions are made, the details ignored in the first stage will then be considered in the second stage. In such a decision making mode, it is important that in the first stage, not only the optimal one is preserved, but the best several as well. Because even if an option is not the best, it might have potential benefits in the second stage and that could make it a better option than the one that was optimal in the first stage. Under such a circumstance, decision makers need the solutions that are "almost the best".

In sum, when decision makers face the real world problems, they need 1) to get the good options at a cost of limited time and computer storage space, 2) to consider several objectives and their trade-offs, and 3) to obtain the solutions that are potentially good. To provide tools for the demands is the main goal I would like to achieve in my work.

1.1.3 Representation of System Architectures

When analyzing a complex system, the decision makers cannot study the system as a whole due to its complexity. They need to decompose the system into manageable parts and then study the connections and interactions between them. Thus, they need the tools to represent relations between parts and to study the structures of the system. This is the goal of the analysis of system architectures, which is defined by E.F.Crawley as "the allocation of physical/informational function to elements of form, and the definition of interfaces among the elements and with the surrounding context"[10].

There are two main categories of such representation. One is to represent the systems in graphs, the other is to represent them in matrices. The graph representation of a simple system is shown in Figure 1-1, and a matrix representation of the same system is shown in Table 1.1.

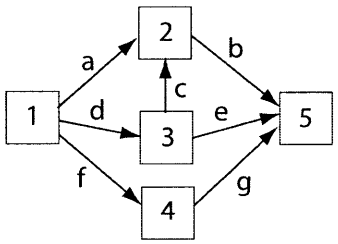


Figure 1-1: An example of system graph.

	1	2	3	4	5
1					
2	✓		✓		
3	✓				
4	✓				
5		✓	✓	✓	

Table 1.1: Matrix representation of the system architecture shown in Figure 1-1

In Figure 1-1, we can see that the system could be decomposed into 5 major parts. Part 1 is connected to parts 2, 3, and 4 with relations a, d, and f, respectively. From the graph, decision makers can easily identify the relations between two parts, and could further infer some valuable information, for example that part 2 needs the input

from part 3, and thus the relation d is better to be analyzed before a.

The matrix representation shown in Table 1.1 is a clear table of the relations between different parts. The check mark indicates the connection between two parts. Although it looks less straightforward than Figure 1-1, it is more clear when the system is so large that the graph representation extends to pages, especially when the different components of the system is highly correlated. The matrix representation is also more convenient when computational analysis of the system is required. A more detailed discussion will be provided in Section 4.1.

Since the two representations of the systems are widely used in system architecture analysis and have complementary advantages and disadvantages, decision makers need theories and methods to bridge the two kinds of representations, so that people can go back and forth and communicate with other system architects. Another goal of this thesis is to establish such a connection and enable people to translate the two representations with ease.

1.1.4 Decomposing and Clustering Complex Systems

A typical process of analyzing a complex system is to first decompose it into small parts that are manageable and analyze them separately, and finally study the integration of all the parts. The question, however, is how to appropriately decompose the system. A widely used approach is to first thoroughly decompose the whole system into "atoms", and then group them into clusters. The analysis will be based on the clusters instead of the atoms. To do such a clustering, matrix representation of the system is advantageous. Figure 1-2 shows a simple example.

When we want to analyze the structure of a bicycle, we first thoroughly decompose it into the simple mechanical parts, i.e., pedals, chain, wheels, etc.. Then we can put a check mark (the number "1" in the matrix) in the entry if the two parts are related. Based on the matrix, we can cluster the parts. There are some rules to follow: 1) we want the parts in a cluster to be closely related, 2) the parts in different clusters should not have a strong connection, and 3) the number of total clusters should be sufficiently large, so that the decomposition is effective. The thick borders in Figure

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

Figure 1-2: An example of DSM clustering [31]. The thick lines indicate the clusters.

1-2 indicates a "good" clustering.

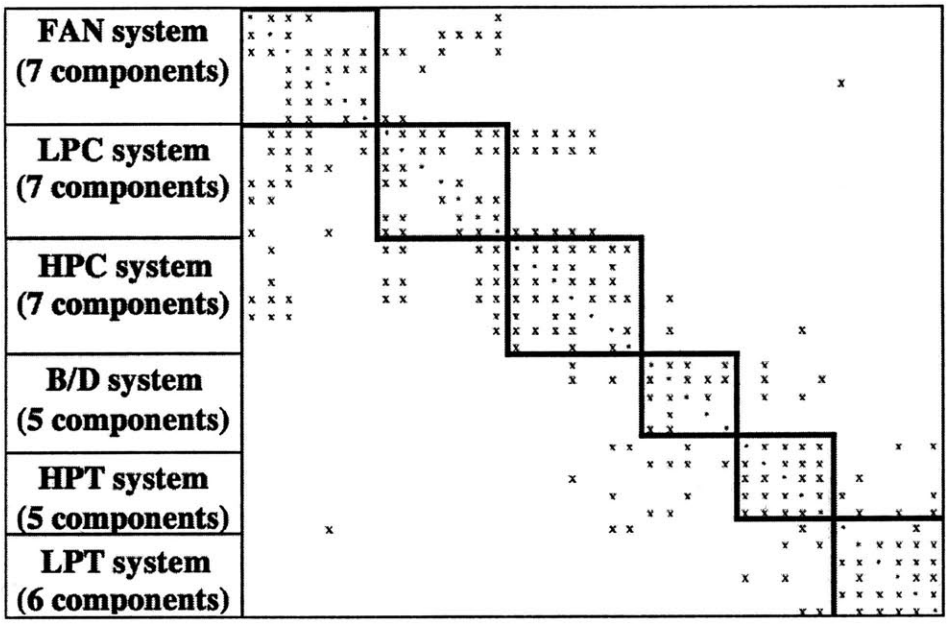


Figure 1-3: An example of real-world DSM clustering problem – A jet engine design problem [37].

The system could be as simple as the one shown in Figure 1-2, where there are only 9 components, and could be as complex as the one shown in Figure 1-3 (refer to [37] for details), where 37 components are included in the DSM. Analysts of complex systems need theories and automatic tools that can help them cluster the parts effectively and efficiently. Thus, another goal I would like to achieve is to provide an effective method for system architects to decompose and cluster complex systems in practice.

1.1.5 A Brief Summary of Motivations and Objectives

In sum, there are three main objectives corresponding to three main motivations in this thesis:

1. Find the good options in complex and large-scale systems for decision makers based on multiple objectives in limited time and storage space.

2. Establish a bridge between the two main representation methods in system architecture: the graph representation and matrix representation. Reveal the inherent relationship between the two representations.

3. Develop a method that can effectively cluster the elemental parts in complex systems.

The first problem is in the domain of optimal decision making and the second and third requires optimization methods for system architectures. Accordingly, the first problem is considered in Chapter 2, with case studies in Chapter 3, and the last two are discussed in Chapter 4.

1.2 Background

In this section, I will introduce the background knowledge needed in or related to my discussion in this thesis. There are three main topics: first is about how to encode the decision making problem and how to solve it, the second is how to solve a problem that is multi-objective, the last one is about the Design Structure Matrix (DSM) and Object Process Diagram (OPD).

1.2.1 Modeling and Solving Decision-Making Problems as Single-Objective Constraint Optimization Problems (COP)

This subsection discusses two important issues of solving decision-making problems: how to model the decision-making problems and how to get the solutions from the

model. The approach I use is to treat the decision-making problems as Constraint Optimization Problems (COP). The reason is that there are several key things that are common in the decision making problems which matches the COPs very well. I will first give the definition of COP with an example, and then explain why the decision-making problems are similar to COPs.

COP is composed by four fundamental parts:

1. A set of decision variables.

2. A set of options for each decision variable.

3. A set of constraints restricting the options for the decision variables.

4. A set of objective functions expressed by the decision variables.

Table 1.2 shows an example of COP. There are 3 decision variables, A, B and C. Decision variable A has three options, 2, 4, and 7. Similarly, decision variable B has 2 options and C has 4 options. There is a constraint $|B - C| \geq 2$, restricting the freedom of choosing B and C at will. There is a single objective: to maximize the function $-A + 10B + 2C$.

Decision Variables	A, B, C
Options	A = {2, 4, 7} B = {3, 8} C = {2, 5, 6, 9}
Constraint	$\|B - C\| \geq 2$
Objective	max $-A + 10B + 2C$

Table 1.2: An example of Constraint Optimization Problem (COP)

Most of the decision-making problems have the same elemental parts as COP: The decision makers have to make a series of decisions, which correspond to the decision variables; for each decision, there are several options, which correspond to the options for decision variables; there are constraints that limit the choice of the options, and it corresponds to the constraints in COP; and there are always some kinds of goals to achieve or objectives to be optimized, which corresponds to the objective functions.

From the discussion above, we can see that COP is a perfect match with many decision-making problems. I provide two case studies in Chapter 3. The two decision problems are both extracted from practical decision-making problems. It verifies such a claim in some degree.

To solve the COPs, we need to address two main issues, which leads to two classes of algorithms: one is to find the solutions that do not violate the constraints, and the other is to find the best solution. For the first problem, we only need to consider the constraints, without objectives. This class of problems is called Constraint Satisfaction Problems (CSP)[33]. CSP is intensively studied in the Artificial Intelligence (AI) community. The main goal of CSP is to find one solution that satisfies all the constraints, without considering any objective. For the second problem, on the other hand, we search for the best solution, and use the constraints to test the validity of the solutions. Now I will briefly introduce one classical algorithm for each of the two problems. The two algorithms are closely related to the algorithm that I introduce in Chapter 2. A systematical introduction of the algorithms and ideas could be found in [33] and [13].

1. The backtracking algorithm for solving CSPs.

   The basic idea of backtracking algorithm is to assign an option to one decision variable once at a time. Every newly assigned option, however, should not violate any constraint based on the assigned set of variables. When none of the options of the decision variable could satisfy all the constraints, we backtrack, i.e., change the assignment of the last decision variable, and then continue the process until we find one full assignment that satisfies all the constraints, or no assignment could be found.

   To better illustrate the process, let us consider the problem described in Table 1.2 (disregard the objective function). A partial process of assigning options to decision variables is shown in Figure 1-4. (The graph is called a search tree.)

   We first assign one option for decision variable A, i.e., let $A = 2$. Then we assign a value to B. Notice that this assignment should not violate any constraints

given $A = 2$. Since all the options are valid, we arbitrarily assign 3 to B. In the next step, we need to pick a value for variable C. Assume we first let $C = 2$, however, this assignment violates the constraint $|B - C| \geq 2$. Thus, value 2 cannot be assigned to C. We pick another value 5 for C, and we get a valid set of assignment, $A = 2, B = 3, C = 5$.

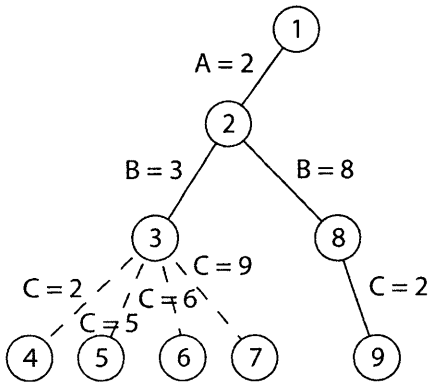


Figure 1-4: Backtracking algorithm for solving single-objective COPs.

To further demonstrate how the backtracking works, assume we have another constraint $|A - C| \leq 1$. Now all the values 5, 6 and 9 for variable C will violate the constraint $|A - C| \leq 1$. Thus, we know the previous partial assignment $A = 2, B = 3$ is not valid, and we have to backtrack. Now we assign value 8 to B and come to node 8. When we try to assign 2 to C, we can see that both constraints $|B - C| \geq 2, |A - C| \leq 1$ are satisfied. Thus, the full assignment $A = 2, B = 8, C = 2$ is a feasible solution.

This is the basic algorithm that is widely used to solve CSPs. There are many techniques such as backtracking with forward tracking, back jumping, etc. that are based on the same thought and improve the overall search efficiency in practice.

Among the techniques, I would like to mention the dynamic ordering of the decision variables. One might have noticed that the sequence of the decision variables that are assigned options is arbitrary in the example. The sequence in the example is A, B then C, but it could also well be C, B then A. In practice, this sequence may affect the efficiency of the algorithm. The dynamic ordering

technique is developed to change such a sequence to reduce the options that are needed to examine. There are different criteria that are adopted. One of them is to assign the one that has the widest connections to other decision variables in the constraints. For example, C is involved in both constraints in the case studied in Figure 1-4, thus, we might want to assign a value to it first to reduce the possible number of options of other decision variables. Another idea is that we can assign an option to the one that is least restricted. It might lead to a feasible solution more quickly. The criterion I use in Chapter 2 is another one which assigns a value to the decision variable that has the least number of options first. The rationale is that if we can eliminate one of the options, a large number of offsprings in that branch will also be eliminated.

In this part, I introduce the background of using backtracking algorithm to solve CSP problems. The algorithm assigns an option to the decision variables while keeping track of the constraints. It backtracks when no feasible option exists for a decision variable. The algorithm seeks only a feasible solution, i.e., it does not consider any objective function. One can always find all feasible solutions and calculate the objective values to pick the best one, but it is not a good strategy. We need an approach that can use the information implied in the objective functions to guide the search. That is, we should be "smarter" when choosing the options for the variables. Therefore, we need algorithms like A*.

2. A* search for COPs

The goal of the A* algorithm for COPs is to find a best solution according to the objective function. There are three important functions in the the algorithm, the function $g(\cdot)$ is the value related to the decisions that have been made, while the function $h(\cdot)$ is the value of the heuristics, which is the "approximating" function of the unassigned decision variables, and function $f(\cdot) = g(\cdot) + h(\cdot)$ is the value that is used as a criterion to choose which option to assign to the next decision variable first. The heuristic function $h(\cdot)$ is the key to the search

efficiency. The closer the value of the function $h(\cdot)$ is to the real value of the unassigned decision variables, the faster the search could reach the best solution.

Let us consider the example shown in Table 1.2 and Figure 1-4 again. The objective function is: $\max -A + 10B + 2C$. Assume we are at step 2 (node 2 in Figure 1-4). Since $A = 2$, if we assign 8 to B, we have $g(\cdot) = -A + 10B = 78$, while if we assign 3 to B, we have $g(\cdot) = -A + 10B = 28$. Let $h(\cdot)$ be the maximum possible value of the unassigned variables. Then we have $h(\cdot) = 2C_{max} = 2 \cdot 9 = 18$. Thus, the function $f(\cdot) = g(\cdot) + h(\cdot) = 78 + 18 = 96$ for the option $B = 8$, and $f(\cdot) = g(\cdot) + h(\cdot) = 28 + 18 = 46$ for the option $B = 3$. Notice that the f value of $B = 8$ is greater than $B = 3$, it means that if we use the heuristic function described above, the assignment $B = 8$ is more promising than $B = 3$ (since we are maximizing the objective value). Thus, we should assign $B = 8$ first. Notice that it does not mean that we will not check $B = 3$, it just means that we will check $B = 8$ first. If it turns out that $B = 8$ is not a valid option, we will backtrack and assign 3 to B again.

In summary, the A* algorithm uses the heuristic function to decide which option to assign to the decision variable first. It keeps track of the values for each node in a tree like the one shown in Figure 1-4, and store them in a queue. The one with the maximum f value will always be checked first.

Theoretically, as long the heuristic function is admissible (the value of the function $f(\cdot)$ is guaranteed to be greater than or equal to the value of the full assignment), the result is optimal. Refer to [14] or [33] for the detailed explanation and proof.

I briefly introduce the algorithms of solving single-objective COPs in this subsection. To solve multi-objective COPs, however, we need more techniques. In the next subsection, I will introduce several widely used multi-objective optimization methods.

1.2.2 Multi-objective Optimization Methods

The multi-objective optimization methods are mainly researched in the operations research community. The book [20] and papers [16][42] provide comprehensive review of the main theories and approaches. Here I would like to introduce only the algorithms that are related to this thesis, including the ones that are used in my work and the ones that are very popular and widely used.

A class of multi-objective optimization methods are based on single-objective optimization approaches. They transform the multi-objective problems into single-objective problems, and then use the single-objective optimization methods to solve them. Two general approaches are adopted to manipulate the multiple objectives: 1) put weights on different objectives, and 2) change the objectives into constraints. The basic idea of the first approach is to put different weights on different objectives (intuitively, give heavy weights to the objectives that people think are more important), and sum the objectives to form a new single objective. There are a variety of extensions of this method under different names like goal programming[41], vector optimization [1][28], etc.. The basic idea of the second approach is to set a tolerance level for all but one objectives, (for example, restrict all but one objectives to be greater than or equal to 0), and optimize the only objective that is left. The first approach is demonstrated in detail in Section 2.3.1, and the second in Section 2.3.2.

Before I introduce the second and third classes of methods, I need to explain the concept of Pareto front. In the multi-objective optimization method I introduce above, only one solution will be obtained. However, most of the time, decision makers need a variety of solutions that are "good" against some objectives. For example, assume that we try to maximize two objectives (o_1, o_2). If the objective values of solution 1 is $(2, 5)$, and the objective values of solution 2 is $(3, 10)$, it is obvious that solution 2 is better. If, however, the objective values of solution 1 is $(3, 5)$, and the objective values of solution 2 is $(2, 10)$, we cannot assert that either solution is better than the other since they have at least one objective value that is greater. Thus, we call one solution dominated by another if and only if all of its objective values are

not better than another. If a solution is not dominated by any other solutions, we call it undominated. All the undominated solutions form the Pareto front. The goal of the following two algorithms is to identify the Pareto front, i.e., to find all the undominated solutions, or so-called Pareto optimal solutions.

The second multi-objective optimization method I want to introduce is very popular in recent years. It is called Evolutionary Multi-Objective (EMO) algorithm[8]. The goal of the EMO algorithm is not to only find one solution, but the whole Pareto front. The algorithm is derived from the general genetic algorithm[27], whose basic idea is to keep a population of solutions and evolve them by the crossover and mutation. The idea of keeping a set of solutions fits in the framework of finding a set of solutions on the Pareto front, and turns out to be effective.

The first effort of extending the genetic algorithm into the multi-objective area is made by Schaffer [34] in 1984. The basic idea is just to keep a population, evaluate each objective of them, then generate the next generation by shuffling the population. The shortcoming is noticed by many researchers and the author himself. The algorithm tends to converge to a single point or an area on the Pareto front instead of the whole Pareto front.

The idea of "sharing" is brought up by Goldberg et. al. in [23]. Since the method developed by Schaffer tends to converge into a single point, we want to avoid the assimilation of the solutions and keep the diversity on the Pareto front. They introduced a parameter $\sigma_{sharing}$, and use an degradation approach to reduce the fitness value of the individuals in a crowded area. For example, in Figure 1-5, the candidate 1 is in a crowded neighbourhood, thus, to maintain the diversity, candidate 2 has a better chance to be kept into the next generation.

There are two different implementation of this suggestion: Fonseca and Fleming[21] used the ranking method. A point's ranking is determined by $\text{rank}(x_i, t) = 1 + p_i^{(t)}$, meaning 1 plus the number of points dominating the point x_i in the t'th generation. Figure 1-6 shows an example.

Then the best fitness value is assigned to different points by the different levels. They also have a technical way of choosing the parameter $\sigma_{sharing}$. Refer to [21] for

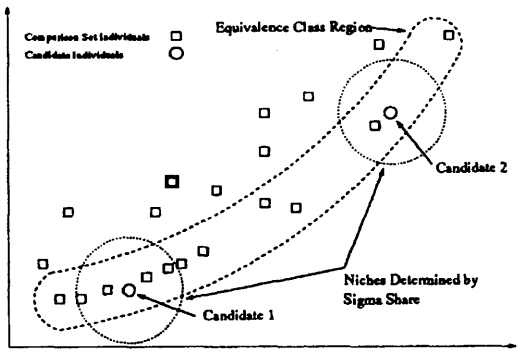


Figure 1-5: The idea of "sharing" used in Evolutionary Multi-Objective (EMO) algorithms.

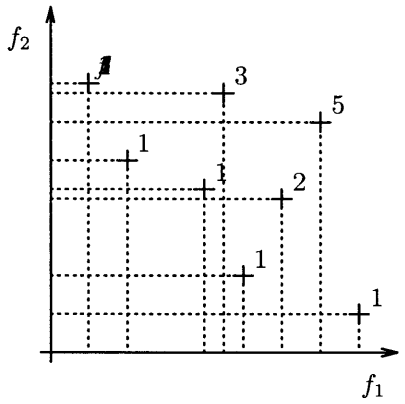


Figure 1-6: Multi-objective ranking. The value of each solution is determined by the solutions it dominates (including the solution itself). The solutions with greater value have better chances to survive the selection in the Genetic Algorithm.

details.

Another approach called "Pareto domination tournaments" is proposed by Horn etc. in [24]. The basic idea is that when choosing the next generation of population, we do not compare two individuals directly, but compare them with a randomly chosen group, say 10%, from the whole population. If one is dominated by the group and the other is not, then the undominated one is chosen. If both dominated or both undominated, the number of the individuals near the candidates are calculated: $m_i = \sum_{j \in Population} Sh(d[i,j])$, where $Sh(d)$ is a decreasing function of $d[i,j]$, such that $Sh(0) = 1$, and $Sh(d \geq \sigma_{share}) = 0$. Then there could be several different ways of determining $Sh(d)$ where $d \in [\sigma_{share}, 1]$, like $Sh(d) = 1 - d/\sigma_{share}$ proposed by [24]

or $Sh(d) = 1 - (d/\sigma_{share})^2$ proposed by [38]. The fitness function is then degraded to f_i/m_i, which means the individual in a crowd gets slimmer odds to survive.

The paper [38] provides general comments on the methods mentioned above and gave simulation results. Another recent paper [8] provides updates in the recent EMO research: Elitism, the use of an external population (secondary population) is used in the next generation of EMO.

The last multi-objective optimization method I want to introduce is the multi-objective A* algorithm[40]. It is a multi-objective version of the generic A* algorithm. The idea of the multi-objective A* algorithm is that instead of choosing the option that has the best objective value and expand that branch first in the search tree, we expand all the branches that have the undominated objective values. Let us consider the shortest path searching problem shown in Figure 1-7 as example.

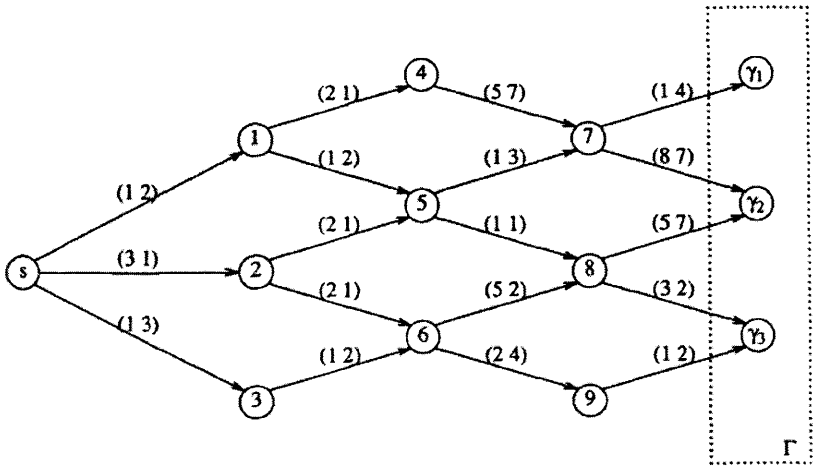


Figure 1-7: Multi-objective A* algorithm [40]. The algorithm examines and expands the nodes with undominated heuristic values first.

There are two numbers on each of the arcs, representing the two objective values of choosing the next node. The goal of this problem is to find the path with the minimum cost from s to $\gamma_1 - \gamma_3$. Assume the heuristic function is the costs on the arc. At the first step, we start from node s and have 3 options – nodes 1 to 3. Notice that the option 3 with cost (1,3) is dominated by option 1, (1,2). Thus, we first examine nodes 1 and 2. With the cost from s to node 4 being (3.3), to node 5 through node 1 being (2,4), to node 5 through node 2 being (5,2), and to node 6

being (5,2), we should check the paths s-2-5, s-2-6 and s-3 first, since the cost from s to 3 is (1,3) which dominates costs (3,3) and (2,4). We can keep expanding the search tree in the same way until we reach one of the γ nodes.

Stewart, B.S. and White III proved that the algorithm is sound and complete in [40], and gave a few more examples in their paper. The algorithm can find all the solutions that are undominated.

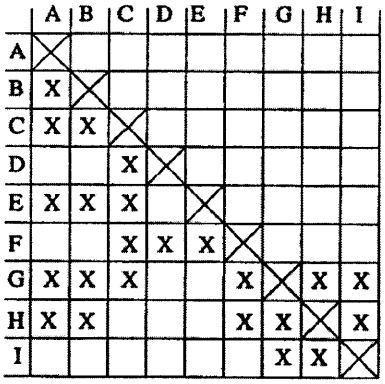
In sum, I introduce three multi-objective optimization techniques above, including 1) changing multi-objective problems into single-objective ones, 2) evolutionary multi-objective algorithm, and 3) multi-objective A*. In Chapter 2, I develop an algorithm based on the weighted sum approach and compare it with the multi-objective A* algorithm.

1.2.3 Design Structure Matrix (DSM) and Object Process Diagram (OPD)

As mentioned in Section 1.1.3, there are two representations that are mainly used in system architecture design. One is the matrix representation, and another is the graph representation. The most widely used matrix representation is Design Structure Matrix (DSM), while an important graph representation is Object Process Diagram (OPD). In this subsection, I briefly introduce the two representation methods.

DSM is developed to manage the product development process, which was a new ground for competitive advantage in manufacture firms back in 1980s [3]. Steward developed the Design Structure Matrix (DSM) as a tool to identify the dependencies between the tasks and to sequence the development process [39]. In this matrix, a task is assigned to a row and a corresponding column. Reading down a column reveals which tasks receive information from the task corresponding to the column. Reading across a row reveals all the tasks whose information is required to perform the task corresponding to the row. Figure 1-8 shows an example of DSM.

The use of DSMs in both research and industrial practice increased greatly in the 1990s. DSMs have been applied to the building construction, semiconductor, automo-



	A	B	C	D	E	F	G	H	I
A									
B	X								
C	X	X							
D			X						
E	X	X	X						
F			X	X	X				
G	X	X	X			X		X	X
H	X	X				X	X		X
I							X	X	

Figure 1-8: An example of Design Structure Matrix (DSM) [3]

tive, photographic, aerospace, telecom, small-scale manufacturing, factory equipment, and electronics industries, etc.[2]. System engineers use it to represent architectural components and interfaces. Organization designers use it to document communication networks. Economists summarize the effects of a change in one product's attributes on other products in a matrix. In decision making processes, DSM is useful when the decision maker need to analyze the constraint relationships between the components.

OPD, on the other hand, is a graphical representation of systems. It is mainly developed by Dov Dori [15] as an important part of the object-process methodology. OPD includes a clear and concise set of symbols that form a language enabling the expression of the system's building blocks and how they relate to each other. It is a symbolic representation of the objects in a system and the processes they enable.

OPD represents the two things that are inherent in a system: the objects and the processes. This duality is recognized throughout the community that studies systems, and sometimes goes by labels such as form/function, structure/function, and functional requirements/design parameters. Objects are what a system or product is, and processes are what a system does. Having the two elemental components in a system, OPD allows a clear representation of many important features of a system: its topological connections, its decomposition into elements and sub-elements, the interfaces among elements, and the emergence of function from elements. OPD also enables decision makers or system analysts to mark the special relations between objects and processes, which is absent from the traditional network-like graphs. Figure 1-9 shows

an example of OPD.

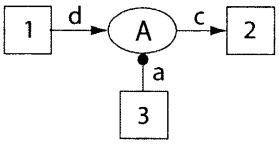


Figure 1-9: An example of Object-Process Diagram (OPD). The rectangles represent objects, and the oval represents a process.

In the OPD shown in Figure 1-9, the rectangles represent the objects and the oval represents the process. Object 1 is the input. Going through process A which has 3 as an instrument, object 1 becomes 3, which is an output. This is called a canonical case, because it matches a natural sentence: 1 is the subject, 2 is the object and A is the predicate. 3 is the adverb which decorates the predicate.

In this subsection, I briefly go through the two useful representation methods in system architecture: DSM as a matrix representation and OPD as a graphical representation. I discuss them in detail in Section 4.1, where I further describe their characteristics.

1.3 Specific Objectives

In this section, I introduce the specific objectives I want to achieve in this thesis. There are three main objectives: 1) to model decision-making problems as multi-objective COPs and to solve them efficiently, 2) to study the projection relation between OPDs and DSMs, and 3) to develop a method to solve the single and multiple DSM clustering problem.

1.3.1 Modeling and Solving Decision-making Problems as Multi-objective COPs

The first objective of my work is to develop a way to model the decision-making problems as multi-objective COPs. As mentioned in Section 1.2.1, many decision-making problems have a similar structure as multi-objective COPs, that is, 1) a set

of decisions needed to be made, 2) several options for each decisions, 3) a set of constraints restricting the relations between the decisions, and 4) a set of objectives to be optimized. In order to encode the practical problems, I need to develop a prototype of modeling language that has the four parts.

Having the modeling tool at hand, I have another objective: to solve the problem described by the modeling language. More specifically, there are two main tasks: the first is to develop an algorithm that solves the single-objective problems, and the second is to extend the algorithm so that it also solves the multi-objective COPs. Also, based on the robustness consideration, I also need to provide an approach that can find not only the solutions on the Pareto front, but also the ones in an area that is close to the Pareto front. I also want to make it possible for decision makers to adjust the size and shape of such an area.

1.3.2 Connections between OPD and DSM

As mentioned in Section 1.1.3, system architects need a bridge between the graph representation and matrix representation. Since OPD and DSM are two important tools in each of the representation methods, I will try to reveal the inherent relation between the two representation methods and establish a mapping from OPD to DSM, and from DSM to OPD as well. In addition, since we expect such a translation happens frequently, we need to enhance the overall efficiency of the translation. Another goal is to fully exploit the features of the two methods and condense the time and space needed to translate back and forth between them.

1.3.3 Single/Multiple DSM Clustering

Another objective of my work is to develop a method to provide good clusterings for DSMs. The goal of such clusterings is to divide the components of the DSMs into several groups which have as little connection with other groups as possible. The clustering method should also be able to put the components that are closely related to each other in the same cluster. Moreover, decision makers usually need more

than one possible clusterings and need to choose the one that matches the reality the best. The DSM clustering method should also be flexible enough to provide different clusterings according to the different demand of the decision makers.

Under some circumstances, there could be more than one DSMs with the same set of components. The system architects need to cluster the components while considering all the DSMs simultaneously. To be more specific, one clustering might be ideal for one DSM, but awful for another. Thus, when clustering multiple DSMs at the same time, one need to consider the trade-offs for all the DSMs. As the single DSM clustering is already a hard task for system architects, one can imagine how hard it could be to cluster several ones at the same time. Thus, another objective of my work is to extend the single DSM clustering approach to multiple DSM clustering, and to offer decision makers the clusterings that achieve a good balance between all the DSMs.

1.4 Summary and Synopsis

In this chapter, I start with the motivations and general objectives of my work that is summarized in this thesis. Generally, decision makers and system architects need tools to model and analyze the complex systems. They also need tools to help them make decisions based on the model extracted from the real problems. During the process of analyzing complex systems, they need graphical tools and matrix tools to identify and represent the connections and interactions between different components in the system. A translating method between the graphical and matrix tools is necessary for them to use both tools freely. Another issue that is needed to be addressed is the decomposition of a large-scale system. Since system architects need to break a complex system down into several manageable parts, they need tools to help them decide a reasonable way of such a decomposition. All the demand mentioned above leads to the general objectives of my work.

I then introduce the background that is either the foundation of my work, or used as a comparison, or closely related to my work. Corresponding to the motivations

and objectives, the background introduction includes the approach that can be used to model the decision-making problems, the algorithms that could be used to solve the problems, and the concept of DSM and OPD.

In the third part, I further explain the specific objectives: to develop a method that can be used to model the practical problems as COPs, to research sound and complete algorithms that could solve the multi-objective COPs effectively, to reveal the relation between DSM and OPD and provide an approach to transform between the two representations, and to provide an effective way that could cluster single DSMs as well as multiple DSMs.

The thesis is organized corresponding to the objectives. Chapter 2 is a complete explanation of the approach of modeling and solving the decision problems as COPs. Chapter 3 provides two case studies as the application of the modeling technique introduced in Chapter 2. It also provides the results solved by the newly developed algorithms. In Chapter 4, I first discuss the connection between DSM and OPD, and then describe an optimization model that can solve the single DSM clustering problem. I finally extend the model to solve multiple DSM clustering problems. In the last Chapter, I summarize the contribution and provide some ideas about the future work.

Chapter 2

Theory of Optimal Decision Making

The main goal of this chapter is to provide an effective way to solve decision-making problems in complex systems. To solve a decision-making problem in a complex system, there are generally two major steps: 1) to encode the problem in a model, and 2) to solve the model and get the good decisions. Corresponding to the two major steps, this chapter consists of two major parts: how to model the decision-making problems, and how to solve the model and find the good decisions.

In the first part, I explain why many decision-making problems can be modelled as multi-objective Constraint Optimization Problems (COP), and how to model decision-making problems as COPs. In the second part, I first introduce the Conflict Directed A* (CDA*) algorithm which solves single-objective COP in Section 2.2. Then in Section 2.3, I introduce three multi-objective optimization methods which can make use of the algorithm that solves single-objective COPs to find the Pareto front of the multi-objective COPs. In Section 2.4, which is the core section of this part, I develop a new algorithm based on CDA* and multi-objective optimization methods. The new algorithm can find all solutions on the convex Pareto front. Then I extend it to find solutions in a region that is close to the convex Pareto front. I also propose an improved algorithm which finds all the solutions on the complete Pareto front. I close this chapter with a summary of the two major parts.

2.1 Modelling Decision-Making Problems as Multi-objective COPs

This section explains why we can model decision-making problems as multi-objective COPs and how to model them as COPs. I first introduce the definition of multi-objective Constraint Optimization Problem (COP) and illustrate its similarity to the real-world decision-making problems. Such similarity enables us to encode many decision-making problems as COPs. In the second subsection, I mainly introduce how to model the decision-making problems as COPs. I illustrate how to model variables, domains, constraints and objectives in COPs for decision-making problems. Finally, I summarize this section with some comments.

2.1.1 Constraint Optimization Problems (COP) and Decision-making Problems

In this subsection, I introduce what is multi-objective Constraint Optimization Problem (COP) and why multi-objective COPs are similar to many decision-making problems. First, I introduce the definition of Constraint Satisfaction Problem (CSP)[33], and then I introduce the definition of COP on top of CSP. CSP can be expressed as a triple $(\mathbf{x}, \mathbf{D_x}, C_x)$. $\mathbf{x}$ is a set of variables $x_i \in \mathbf{x}$ that ranges over finite domain $D_{x_i} \in \mathbf{D_x}$. $C_\mathbf{x}$ denotes a set of constraints. Formally $C_\mathbf{x} : \mathbf{x} \rightarrow \{true, false\}$. The goal of a CSP is to find a $\mathbf{x_s}$ that satisfies all the constraints, i.e., $C_\mathbf{x}(x_s) = true$. A multi-objective Constraint Optimization Problem (COP) can be defined as (CSP, $\mathbf{g}$), or $(\mathbf{x}, \mathbf{D_x}, C_x, \mathbf{g})$, where $\mathbf{g}$ is a set of objective functions consisting of functions $g_i : \mathbf{x} \rightarrow \mathbb{R}$.

A solution is defined as a full assignment to $\mathbf{x}$ such that all constraints are satisfied, i.e., $C_x(\mathbf{x}) = true$. A solution $\mathbf{x}$ is dominated by another solution $\mathbf{x}'$ if $\forall k,\ g_k(\mathbf{x}) \leq g_k(\mathbf{x}')$ (assume we are maximizing all g_i). In other words, a solution is dominated by another solution if none of its objective value is better than the other's. A solution that is not dominated by any other solution is called an un-

dominated solution, or a Pareto optimal solution. All the undominated solutions form the complete Pareto front. A solution that is not dominated by any convex combination of other solutions is called a convex undominated solution, or a convex Pareto optimal solution. Formally, a solution $\mathbf{x}$ is convex Pareto optimal if and only if $\forall n \geq 1, \forall \lambda_{1..n}, \forall \mathbf{x}_{1..n}, \exists k \;\; s.t. \; g_k(x_i) > \sum_{j=1}^{n} \lambda_j \cdot g_k(\mathbf{x}_j)$, where $\sum_{j=1}^{n} \lambda_j = 1, \; \lambda_j \geq 0$. All the convex undominated solutions form the convex Pareto front.

The goal of solving a single-objective COP is to find one or several solutions that have the best objective value. Sometimes we also need the second-best, third-best, etc. solutions, until a certain number of top solutions are obtained, or the objective value goes down beyond a threshold. The goal of solving a multi-objective is either to find the complete Pareto front or to find the convex Pareto front.

In sum, a COP is composed of four parts: a set of variables, a finite domain for each variable, a set of constraints, and a set of objective functions. Table 2.1 shows an example of COP.

Variables	A, B, C
Domains	$A = \{2, 4, 7\}$ $B = \{3, 8\}$ $C = \{2, 5, 6, 9\}$
Constraint	$\|B - C\| \geq 2$
Objectives	max $-A + 10B + 2C$ max $7A - 5B - 2C$

Table 2.1: An example of multi-objective Constraint Optimization Problem (COP)

There are 3 variables A, B and C, that is, $\mathbf{x} = \{A, B, C\}$. The domain for variable A is $\{2, 4, 7\}$, or formally, $D_A = \{2, 4, 7\}$. Similarly, $D_B = \{3, 8\}$ and $D_c = \{2, 5, 6, 9\}$. The only constraint is $|B-C| \geq 2$. If $A = 4, B = 8, C = 2$, we have $C_x(\mathbf{x}) = true$ since the assignment satisfies the constraint. On the other hand, if $A = 4, B = 8, C = 9$, we have $C_x(\mathbf{x}) = false$, since the assignment violates the constraint. The function set $\mathbf{g}$ consist of two objectives $g_1 = -A + 10B + 2C$ and $g_2 = 7A - 5B - 2C$.

The assignment $A = 4, B = 8, C = 2$ is a solution since the constraint is satisfied, but $A = 4, B = 8, C = 9$ is not a solution since the constraint is violated. Figure 2-1 shows the objective values of all solutions (the red points). The X axis represents

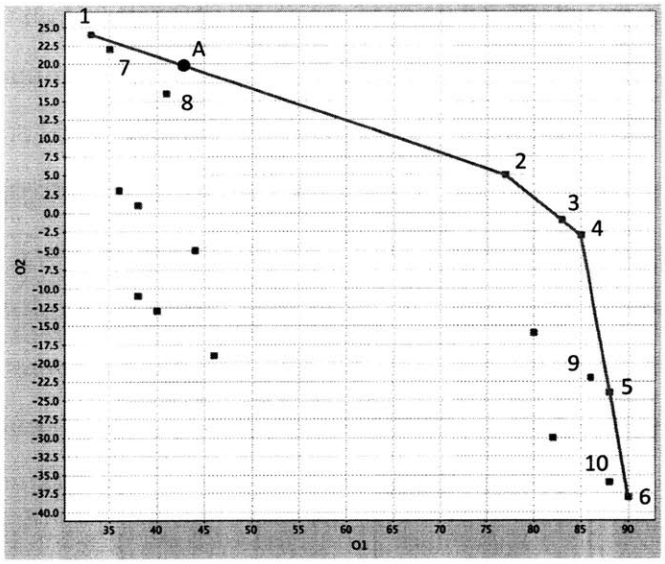


Figure 2-1: The plot of the solutions of the multi-objective COP shown in Table 2.1. Points 1 to 10 are Pareto optimal, while Points 1 to 6 are convex Pareto optimal. The blue line indicates the convex Pareto front.

the value of g_1, and the Y axis represents the value of g_2. The points 1 to 10 marked on the figure are undominated by any other solution. These points form the Pareto front. On the other hand, points 7 to 10 are not convex undominated. Point 8, for example, is dominated by point A on the line connecting points 1 and 2. It means that point 8 is dominated by a convex combination of points 1 and 2. The blue line connecting all the convex undominated points represents the convex Pareto front.

Many decision-making problems are similar to COPs in the sense that they are composed of the same elemental parts: a set of decision variables, a finite domain for each variable, a set of constraints that bounds the variables, and several objectives needed to be optimized. More specifically, people usually need to make a series of decisions when dealing with a decision-making problem. For each decision, the decision makers often have several options, and they need to choose one among these options. These decisions are usually connected with each other, or in other words, a choice of one decision might make some options for other decisions infeasible. Also, the decision makers often have several objectives to optimize, and some of these objectives are conflict with each other in some degree so that the decision makers need to evaluate the trade-offs between them. Finally, the decision makers might

need all the undominated solutions, or even the ones that are dominated, but close enough to the Pareto front. They need these solutions to make the final decision that achieves the best balance between different objectives.

For example, when we want to choose components to assemble a computer, we need to make a series of decisions like what CPU to choose, what motherboard, memory chips, graphical card, LCD, etc. to choose. For each of them, we have finite choices on the market. We also need to consider several constraints like the motherboard we choose should support the CPU, the graphical card should match the LCD, etc.. Moreover, we have two objectives to optimized: One is the performance of the assembled computer, the other is the total cost. We might not need the slowest computer which might also be the cheapest, and we might not be able to afford to buy all the best components. A trade-off, or so-called performance/price ratio should be considered, but just among the Pareto optimal solutions.

Not only the example above, but also many other decision-making problems ranging from simple daily decision-making problems to difficult decision-making in large projects in the real world, can be divided into the four elements. Chapter 3 provides two case studies using the multi-objective COP to model difficult decision-making problems. Since many of the decision-making problems have the same structure as COPs, we can follow the two steps mentioned in the beginning of this chapter to solve them: first model them as COPs and then solve the COPs.

In this subsection, I introduce what is multi-objective COP, and then discuss the structure of real-world decision-making problems and show that they have the similar structures as multi-objective COPs. The conclusion is that we CAN model many decision-making problems as COPs. In the next subsection, I will illustrate HOW to model the decision-making problems as COPs.

2.1.2 Modelling Decision-making Problems as COPs

This subsection discusses how to model decision-making problems in COPs. As mentioned in the previous subsection, many real-world decision-making problems have the similar structure as COPs. They are composed of four parts: decision variables, finite

domain for each variable, constraints and objectives. In this subsection, I illustrate how to express these four parts of decision-making problems respectively.

1. Domains. We can define the domains in lists. For example, consider the case shown in Table 2.2 (The table is a duplication of Table 2.2). We can define the domains as follows:

$$D_a = (2\ \ 4\ \ 7) \qquad D_b = (3\ \ 8) \qquad D_c = (2\ \ 5\ \ 6\ \ 9)$$

The three domains are for decision variables A, B and C, respectively.

Variables	A, B, C
Domains	$A = \{2, 4, 7\}$ $B = \{3, 8\}$ $C = \{2, 5, 6, 9\}$
Constraint	$\lvert B - C \rvert \geq 2$
Objectives	max $-A + 10B + 2C$ max $7A - 5B - 2C$

Table 2.2: An example of multi-objective COP (copied from Table 2.1).

2. Variables. We can define the variables by their names and link them to their domains defined above. For example, we can define the decision variables A, B and C as follows:

$$A : D_a \qquad B : D_b \qquad C : D_c$$

The reason that we define domains and variables separately, but not to define the domains directly with the variables, is that sometimes some of the variables have the same domain, and these variables can share domain definition. For instance, if we need to decide the order of doing things 1 to 5, we can have five decision variables s_1 to s_5, representing the order of doing things 1 to 5. The domains of s_1 to s_5 are all $D_s = (1\ \ 2\ \ 3\ \ 4\ \ 5)$, which means the five things could be either of orders 1 to 5. It saves us the trouble defining five same domains separately for the five decision variables.

3. Constraints. Following the convention of the constraint definition in the AI community, I adopt the Conjunctive Normal Form (CNF) to express the constraints. CNF is a conjunction of clauses, which is a disjunction of literals. A literal is an assertion like $A = 2$ or its negation, NOT $A = 2$. A literal can be either true or false. A clause is a disjunction (the "OR" relationship) of literals. For example, $(A = 2)$ OR (NOT $C = 2$) is a clause. CNF is a conjunction (the "AND" relationship) of clauses. For example, The constraint $|B - C| \geq 2$ can be expressed in the following CNF:

$$\begin{aligned} & ((\text{NOT } A = 2) \text{ OR } (\text{NOT } C = 2)) \\ \text{AND } & ((\text{NOT } A = 4) \text{ OR } (\text{NOT } C = 5)) \\ \text{AND } & ((\text{NOT } A = 7) \text{ OR } (\text{NOT } C = 6)) \end{aligned}$$

The three clauses enumerate all possible combination of B and C that violates constraint $|B - C| \geq 2$. CNF might not be the most efficient way of expressing constraints, but it is always able to encode the constraints in COPs since the domains for decision variables are finite. We can always enumerate every possible violation to the constraints and use "AND" to conjunct them together. CNF is also widely used in the AI community and can be understood and solved by many COP solvers.

4. Objective functions. I use valued constraints to model the objective functions in COPs. To illustrate how to model the objective functions with valued constraints, I 1) explain what is a Mutual Preferential Independence (MPI) property of an objective function, 2) explain what is valued constraint and show how to use the valued constraints to model an objective function with MPI property, and 3) show how to use value tuples to model an objective function without MPI property.

   An objective function g has MPI property when for any variable x that appears in function g, the preference between the assignments to x is independent of the other variables that appear in function g. More precisely, let $\bar{\mathbf{x}}$ be the

other variables that appear in function g, let $x^{(i)}$ and $x^{(j)}$ be any two possible assignments to the variable x, and assume we want to maximize the value of function g. If for one assignment $\bar{\mathbf{x}}^{(\mathbf{k})}$ to $\bar{\mathbf{x}}$, we have $g(x^{(i)}, \bar{\mathbf{x}}^{(\mathbf{k})}) > g(x^{(j)}, \bar{\mathbf{x}}^{(\mathbf{k})})$, then for a function g that has the MPI property, we have: for any assignment $\bar{\mathbf{x}}^{(\mathbf{l})}$ to $\bar{\mathbf{x}}$, $g(x^{(i)}, \bar{\mathbf{x}}^{(\mathbf{l})}) > g(x^{(j)}, \bar{\mathbf{x}}^{(\mathbf{l})})$

The two objective functions shown in Table 2.2 both have MPI property. For example, the preference of the assignment to variable A is independent of the assignments to variables B and C. More specifically, the function value of $g_1 = -A + 10B + 2C$ is $g_1 = -2 + 10B + 2C$ when 2 is assigned to A, and $g_1' = -7 + 10B + 2C$ when 7 is assigned to A. Since $-2+10B+2C > -7+10B+2C$ no matter what are the assignments to variables B and C, we know that assigning value 2 to A is always preferred to assigning 7 to A, or in other words, the preference of assignment to A is independent of the assignments to B and C.

Now I can illustrate what are valued constraints and how to use valued constraints to model objective functions with MPI property. A valued constraint $f : x \rightarrow \mathbb{R}$ is a mapping from an assignment of a variable to a value. An objective function is the sum of the values of the variables, i.e., $g(\mathbf{x}) = \sum_{x_i \in \mathbf{x}} f(x_i)$. For example, in the objective function $-A + 10B + 2C$, assigning 2 to A makes the only term that is related to A, that is, $-A$, to be -2. Thus, the mapping from the assignment of A to the value is $f_A(A) = -A$. Similarly, the valued constraints for B and C are $f_B(B) = 10B$ and $f_C(C) = 2C$, respectively, and the objective function $g_1 = f_A + f_B + f_C$.

The mapping of the valued constraints can be expressed in pairs (Assignment, Value). For example, there are three pairs for the valued constraints corresponding to variables A are $(A = 2,\ -2), (A = 4,\ -4)$ and $(A = 7,\ -7)$, which means when 2 is assigned to A, its corresponding value is -2, and when 4 and 7 are assigned to A, the values are -4 and -7, respectively. Then the complete list of valued constraints for variable A is $((A = 2,\ -2)\ \ (A = 4,\ -4)\ \ (A = 7,\ -7))$. Similarly, the valued constraints for variable B and C are $((B = 3,\ 30)\ \ (B =$

8, 80)) and ((C = 2, 4) (C = 5, 10) (C = 6, 12) (C = 9, 18)), respectively.

The valued constraints might not be the most efficient way of modeling objective functions, but is always feasible because COP has a finite number of variables and a finite domain for each variable. It is also very flexible because the objective function could be very complex, or could be a function of non-numeric decision variables. For example, if one of the decision variables has three options high, medium and low, the corresponding values in the objective function could be, for example, 7, 16 and 43. This might not be easily expressed in a quantitative function, but can be easily expressed as valued constraints. The objective functions with MPI property can be either purely a sum or a product of functions of a single variable. For example, $3A + B^2 + e^C$ is a valid function for decision variables A, B and C, since each term involves only one decision variable. But $A \times B + C$ is not, because two variables are involved in the first term. Also, for the pure product forms, $(A + 2) \times \sqrt{B} \times \log(C - 1)$ is valid, while $(A + B) \times C$ is not.

Some real-world problems, however, do not have MPI property, and now I will introduce how to model the objective functions without MPI property. In many cases, the decision making problems that do not have MPI property can more or less be decomposed into several independent parts that do have MPI property. As an example, the objective function: max $\{A \times B + C\}$ could be decomposed into two independent parts, $A \times B$ and C. The two parts are mutually independent, that is, if we substitute $A \times B$ for an artificial variable D and let the domain of D be all the possible values of $A \times B$, the function becomes a standard MPI form $D + C$, yet this substitution does not change the mapping relations of the original function. This motivates us to define the value tuples to bind variables like (A, B) together.

Formally, a value tuple is defined as a mapping $f : \mathbf{y} \rightarrow \mathbb{R}$, where $\mathbf{y} \subset \mathbf{x}$. Let's consider the example shown in Table 2.3. Notice that MPI property does not hold for the (A, B) combination: the maximum value 3 actually comes from a

minimum combination ($A = -3, B = -1$) of A and B. But MPI holds for the tuples (A, B) and (C). Since we have $A = \{-3, 2\}$ and $B = \{-1, 1\}$, $A \times B$ could be 3, -3, -2 or 2, which means $D = \{3, -3, -2, 2\}$. We call the combination (A, B) a value tuple. Using the concept of value tuple, we have $[(A = -3, B = -1), 3], [(A = -3, B = 1), -3], [(A = 2, B = -1), -2], [(A = 2, B = 1), 2]$. The value tuples corresponding to the objective function are:

$$[(A = -3, B = -1), 3],\ [(A = -3, B = 1), -3],\ [(A = 2, B = -1), -2],\ [(A = 2, B = 1), 2]$$

$$[(A = -3, C = 0), 0],\ [(A = 2, C = 0), 0],\ [(A = -3, C = 6), -2],\ [(A = 2, C = 6), 3]$$

for tuples (A, B) and (A, C), respectively.

- Decision variables:
  (a) Decision $A = \{-3, 2\}$
  (b) Decision $B = \{-1, 1\}$
  (c) Decision $C = \{0, 6\}$
- Constraints:
  $B \times C \geq 0$
  $C - A \leq 2$
- Objective function:
  max $\{A \times B + C \ / \ A\}$

Table 2.3: A single-objective COP example for explaining value tuples.

This subsection describes how to model a decision-making problem as multi-objective COP, which is defined as $(\mathbf{x}, \mathbf{D_x}, C_x, \mathbf{g})$, where $\mathbf{x}$ is a set of decision variables, $\mathbf{D_x}$ is a set of domains that correspond to the variables, C_x represents a set of constraints and $\mathbf{g}$ denotes a set of objective functions. Corresponding to the four elements in COP, I show how to encode domains, variables, constraints and objective functions of a decision-making problem. Two case studies are provided in Chapter 3. I successfully use the method introduced in this subsection to model two real-world decision-making problems.

2.1.3 Summary

The definition of COP is introduced with a pedagogical example at the beginning of this section. The similarity of the structure of COP and many decision-making

problems motivates me to use COP to model real-world decision-making problems. In the section part, I describe in detail how to model the decision-making problems as COPs. The method of modelling the real-world problems is verified by the two case studies shown in Chapter 3.

COP is developed base on the widely studied Constraint Satisfaction Problems (CSP)[33]. In fact, COP can be expressed as $(CSP, \mathbf{g})$, where $\mathbf{g}$ is a set of objective function. This expression means the COP consists a CSP with several objective functions. This is the main reason I adopt the convention of the CSP modelling approach, like the Conjunctive Normal Form (CNF) and the valued constraints. The modelling approach has both strength and weakness. The strength is that it could model most problems that have finite decision variables and finite domains. The weakness, as a consequence of the strength, is that the approach might be inefficient when modelling large-scale problems because it might ask decision makers to enumerate the elements of many sizable domains. The weakness can potentially be overcome by a more compact expression of the variables, constraints and objectives. For example, if we can express the constraint that we want to put things 1 to n in order as $(x_i, i = 1..n) \rightarrow seq(1..n)$, which means x_1 to x_n should be assigned different values from 1 to n, it will be very convenient to model some problems. In fact, this is one of the most frequently used constraints in real-world problems. The development of an efficient language and it compiler which translates the language to COP is one of the main goal in the future work.

In the rest of the sections in this chapter, I introduce how to solve COP. As introduced in the beginning of this chapter, there are two major steps of solving the real-world decision problems: 1) to model them as COP, and 2) to solve the COP. Since I have shown how to get the first step done, I will spend the rest of this chapter introducing the relevant algorithms that solve multi-objective COPs, and then develop my new algorithm based on them.

2.2 Conflict-directed A* (CDA*) Algorithm

In the last section, I explained why and how to model decision-making problems as multi-objective COPs. Starting from this section, I will introduce how to solve the multi-objective COPs. Generally, I use a two-level algorithm to solve multi-objective COPs: The top level repeatedly generates a single objective based on the objective function set, and the bottom level takes the single objective and uses an algorithm to solve the single-objective COP. The series of single objectives generated by the top level guarantees that all the solutions on the convex Pareto front will be found.

This section introduces the Conflict-directed A* (CDA*) algorithm[43], which efficiently solves single-objective COPs, while the next section introduces multi-objective optimization methods, which can be used to generate the series of single objectives which lead to all the convex Pareto front solutions. In the section after the two mentioned above, I combine the CDA* and multi-objective optimization methods to develop my algorithm that solves the multi-objective COPs.

I divide this section into two parts: the first introduces the constraint-based A* algorithm, which could find the best solutions to a single-objective COP. The second part explains what are "conflicts" and shows how to use the conflicts learned during the search process to increase the overall search efficiency. In addition, the second subsection also introduces an extension of CDA* that is useful in the discussions later in this chapter.

2.2.1 Constraint-based A* Algorithm for Single-objective COPs

To illustrate the constraint-based A* algorithm for single-objective COPs, I use the single-objective COP in Table 2.4 as an example. After showing the process of solving the single-objective COP, I provide the pseudo code of the algorithm. Figure 2-2 shows the search tree corresponding to the single-objective COP in Table 2.4.

Nodes 1 to 4: Using the dynamic ordering technique (refer to Section 1.2.1 for details), the algorithm assigns a value to B first. The choice is not arbitrary though. Note the objective function has the MPI property (refer to Section 2.1.2 for details),

Decision Variables	A, B, C
Options	A = {2, 4, 7} B = {3, 8} C = {2, 5, 6, 9}
Constraint	$\|B - C\| \geq 2$
Objective	max $-A + 10B + 2C$

Table 2.4: A single-objective COP

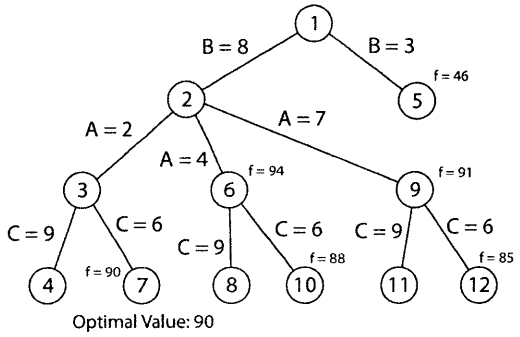


Figure 2-2: Search tree of the single-objective COP shown in Table 2.4

thus, the best assignment for B that maximizes $10B$ is picked, that is, branch $B = 8$ is expanded first. Similarly, we expand $A = 2$ and $C = 9$ in the following two steps.

When Node 4 is reached, each of the variables is assigned a value. Now we need to test whether all constraints are satisfied. If they are, this full assignment is consistent, and the first consistent full assignment found is the optimal solution[43]. If the full assignment is inconsistent, however, we need to continue searching. In the example, $B = 8, C = 9$ violates the constraint $|B - C| \leq 2$, hence we continue searching.

Nodes 5 to 7: The next step is to expand the second best branch for the nodes that are not fully expanded yet. In the example, nodes 1, 2 and 3 are expanded and we get Nodes 5, 6 and 7. To decide which of these node gets to be expanded first, we calculate a bound heuristic on their values (refer to Chapter 1, Section 1.2.1 for details). Because the objective function has the MPI property, a straightforward admissible heuristic is the best objective value of each node while relaxing all constraints. For example, for Node 5, after assigning 3 to B, the best solution for the objective max $-A + 10B + 2C$ is to maximize each term involving unassigned decision variable, i.e., $-A$ and $2C$, respectively. That is, to assign the smallest

value (2) to A and the largest value (9) to C. The corresponding objective value is $-A + 10B + 2C = -2 + 10 \times 3 + 2 \times 9 = 46$. Similarly, for Node 6, after assigning 8 to B and 4 to A, the best choice is to assign 9 to C, which maximizes the term $2C$ and leads to the objective value $-A + 10B + 2C = -4 + 10 \times 8 + 2 \times 9 = 94$. The heuristic value for Node 7 is 90, which is also the objective value for Node 7 since all decision variables are assigned.

Before continuing, note that the heuristic function is admissible. Because the heuristic value is the best value while relaxing all constraints, the best consistent solution of that branch could not exceed the heuristic value when the constraints are enforced. Thus, the first consistent full assignment is guaranteed to be the optimal solutions by the A* algorithm.

Nodes 8 to 12: Now the algorithm picks the node with the highest heuristic value to expand first. Expanding node 6, we get node 8, whose objective value is 94. This full assignment, however, is inconsistent. Hence, Nodes 2 and 6 are further expanded, and Nodes 9 and 10 are obtained. Now Node 9 has the best heuristic value. Thus, the algorithm expands Node 9 and gets Node 11. Again, Node 11 has the best objective value but is inconsistent. Node 9 is then expanded again and Node 12 is obtained. The heuristic value of Node 12 is 85. Now Node 7 has the best heuristic value 90 and will be examined for consistency. This full assignment is consistent and it is the optimal solution with $A = 2, B = 8, C = 6$ and the objective value 90.

In sum, the algorithm keeps choosing an unassigned variable to expand the tree, until all the variables are assigned. When expanding the tree, the branch with the best heuristic value is searched first. If the full assignment passes the consistency test, it is the optimal solution. If it is inconsistent, the algorithm expands the nodes that are not fully expanded yet, calculates their heuristic values and puts them into the priority queue sorted by the heuristic values, then expand the node on the top of the priority queue, i.e, the node with the best heuristic value. Continue until all the decision variables are assigned while testing the consistency after each assignment. Loop until the algorithm finds a consistent full assignment, which is the optimal solution for the single-objective COP, or all assignments are exhausted. The pseudo

code of the algorithm is shown in Algorithm 1.

Algorithm 1 Find the optimal solution for a single-objective COP using the constraint-based A* algorithm

Input: A single-objective COP
Returns: The optimal solution for the input COP

```
Priority queue Q ← {} (Nodes in Q are sorted by their heuristic values)
Evaluate the heuristic value of an empty assignment
Add a node of the empty assignment to Q
while Q is not empty do
  Take the first node N in Q
  if N represents a partial assignment then
    Choose an unassigned variable x with the smallest domain size
    Working domain D' ← D_x (The domain of x)
    repeat
      Assign the best value in D' to x and test the consistency
      if Pass the consistency test then
        Evaluate the heuristic value of the new assignment
        Put a new node of the new assignment in Q
      else
        Take the value just assigned to x out of D'.
      end if
    until A value is assigned to x or D' is empty
  else
    The N is a full consistent assignment. Return this solution.
  end if
end while
No solution is found. Return null.
```

One might have noticed that some subsets of assignments are the source of inconsistency, yet the same sets of assignments are not eliminated in the subsequent search. For example. $B = 8, C = 9$ is the source of inconsistency for Node 4, while it happens again at Node 8. If the algorithm can record the subsets that cause the inconsistency and avoid them after they happen for the first time, we can expect the efficiency of the algorithm to be enhanced. The sets of inconsistent assignments are defined as "conflicts", and the next subsection briefly introduces the way of using them to guide the search.

2.2.2 Conflict Learning and an Extension of CDA*

This subsection explains what are conflicts, how to record the conflicts during the process of expanding the search tree, and how to reuse them to guide the subsequent search. In addition, I introduce an extension of CDA* that is useful in our future discussion.

The last subsection introduces the CDA* algorithm, and shows that some common partial assignments like $B = 8, C = 9$ make several full assignments infeasible. Such a partial assignment is defined as a conflict. More precisely, a conflict is a partial assignment that is inconsistent. Any partial or full assignment that is a super set of any conflict is inconsistent as well. Since a conflict could appear in many full assignments, the overall efficiency could be improved by learning the conflicts from the consistency tests and avoiding them in the future search.

As an example, in Figure 2-3 (a duplication of Figure 2-2, for reader convenience), when the consistency of Node 4 is tested, the conflict $B = 8, C = 9$ is found and recorded. During future tree expansion, when 8 is already assigned to B, there are only three options left for C: {2, 5, 6}. Thus, when expanding node 6, the best branch is $C = 6$ instead of $C = 9$, and we save time and space by not checking and storing Node 8. The time saving effect probably isn't significant in this simple example; however, if there are still hundreds of decision variables waiting to be assigned after A, B, C, the cut of branch $C = 9$ will save much time.

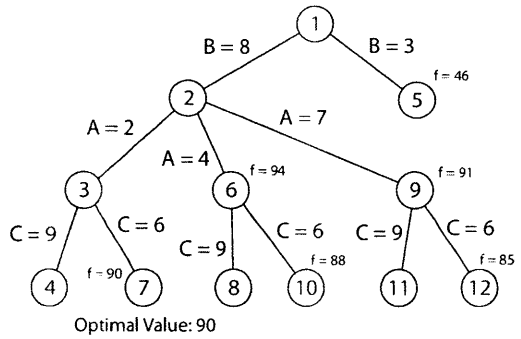


Figure 2-3: Search tree of the single-objective COP shown in Table 2.1

The details of learning and using conflicts to direct the search can be found in

Section 5 of [43]. The conflict learning plays an important role in promoting overall search efficiency in practice. During the execution of my algorithm, the top level generates different single objective functions, and call CDA* to solve the COP with the same set of variables, domains and constraints (only the objective function is altered). Thus, the conflicts learned in previous executions could be reused and will save us a great amount of time. The pseudo code of CDA* shown in Algorithm 2 is revised from constraint-based A* shown in Algorithm 1.

Algorithm 2 Find the optimal solution for a single-objective COP using CDA*

Input: A single-objective COP; Optional: initial conflict set C
Returns: The optimal solution for the input COP; Conflict set C

```
Priority queue Q ← {} (Nodes in Q are sorted by their heuristic values)
Evaluate the heuristic value of an empty assignment
Add a node of the empty assignment to Q
if No initial conflict set C then
  C ← {}
end if
while Q is not empty do
  Take the first node N in Q
  if N represents a partial assignment then
    repeat
      Based on the partial assignment N, choose the next best set of assignments
      that resolves the conflicts in C
      if The new assignment is consistent then
        Evaluate the heuristic value of the new assignment
        Put a new node of the new assignment in Q
      else
        Add the conflict that causes the inconsistency to the set C
      end if
    until A new node is added to Q or no assignment can resolve all the conflicts
    in C
  else
    The N is a full consistent assignment. Return this solution and the conflict
    set C.
  end if
end while
No solution is found. Return null, and the conflict set C.
```

An important generalization of CDA* deserves our attention. If we do not stop

when the optimal solution is found, we can continue to find the second best solution, the third best solution, etc, until a certain number of solutions are obtained, or an objective lower bound is reached. Figure 2-4 shows an example of this generalization for the simple optimal CSP. As discussed later, this generalization will enable us to enumerate the best design to a user.

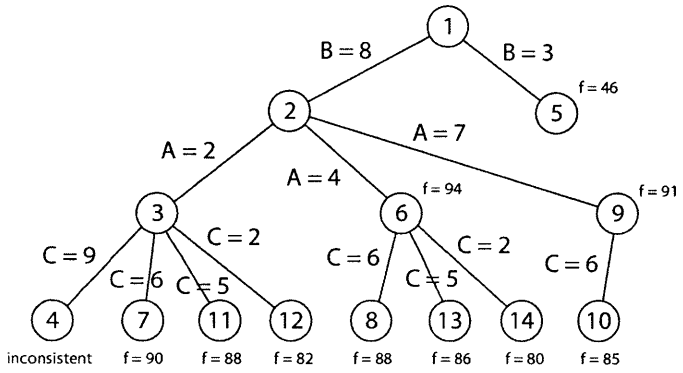


Figure 2-4: Search tree of the single-objective COP shown in Table 2.1 using the generalized CDA* algorithm

Let us assume the cutoff value is 4, or in other words, the minimum acceptable objective value is 90 - 4 = 86. After getting Node 7 as the optimal solution, we continue to explore the tree for the second best solution. At this stage, Nodes 1 to 10 are expanded but Nodes 8 and 10 are not examined yet. Now the algorithm expands Node 3, since its children could have better objective values than Node 8. It turns out that Node 11 has the same objective value as Node 8. Passing the consistency test, these two solutions are the second best. Because the objective values hasn't dropped beyond the lower bound, the process continues: Nodes 3 and 6 are expanded and Nodes 12 and 13 are obtained. Now Node 13 has the best heuristic value. Since the full assignment corresponding to Node 13 is consistent, it is the third best solution.

Next, because the objective value of Node 12 is smaller than the lower bound, this node will not be visited any more. Now Node 13 is the best consistent solution which still stays above the lower bound. Hence, we need to examine Node 13. It is also consistent, and is therefore the fourth best. Now the objective value has dropped to 86, which is still in the acceptable range. Hence, we need to continue. Since expanding

Node 6 could possibly give us a node with a heuristic value that is greater than 85, we have to expand Node 6 now. Node 14 is obtained but has the value 80, which is smaller than 86. Now all the unexpanded or unexamined nodes are with the heuristic values smaller than 86. Thus, we can assert that all solutions with objective values better than or equal to 86 are found and sorted in descending order. The solutions correspond to Nodes 7, 8, 11, and 13. The pseudo code of this generalization is shown in Algorithm 3.

The ability to find "good" solutions other than the optimal one is often important for decision makers. The reason is that when considering a complex decision-making problem, it is ofter not realistic to consider ten objectives at the same time. People focus on two or three most important objectives first, and evaluate the others based on the solutions obtained by optimizing the most important ones. Thus, the best solution to one or two objectives are not necessarily global optimal. Some of the "good" solutions might be better on other objectives and preferred when all the objectives are considered.

This section first briefly introduces how to use conflicts to guide the search. Then an important generalization that enables CDA* to find top solutions other than the optimal one is introduced. The CDA* is used as the bottom-level algorithm in my two-level algorithm introduced in the beginning of this whole section. The next section introduces the multi-objective methods that are useful to build the top-level algorithm.

2.3 Multi-objective Optimization Methods

This section introduces several multi-objective optimization methods that are useful to build my own algorithm. As mentioned in Section 2.2, I use a two-level algorithm to solve the multi-objective COPs: The top level generates a single objective based on the objective function set repeatedly, and the bottom level takes the single objective and uses CDA* (introduced in Section 2.2) to solve the COP. The series of single objectives generated by the top level guarantees that all the solutions on the convex

Algorithm 3 Find top solutions of a single-objective COP using generalized CDA*

Input: A single-objective COP, a cutoff value c; Optional: initial conflict set C
Returns: The top solutions whose objective values are better than the lower bound; The conflict set C

```
Priority queue Q ← {} (Nodes in Q are sorted by their heuristic values)
Evaluate the heuristic value of an empty assignment
Add a node of the empty assignment to Q
Solution set S ← {}
Lower bound bleftarrow − ∞
if No initial conflict set C then
  C ← {}
end if
while Q is not empty do
  Take the first node N in Q
  if N represents a partial assignment then
    repeat
      Based on the partial assignment N, choose the next best set of assignments
      that resolves the conflicts in C
      if The new assignment is consistent then
        Evaluate the heuristic value of the new assignment
        if The heuristic value is no less than the bound b then
          Put a new node of the new assignment in Q
        end if
      else
        Add the conflict that causes the inconsistency to the set C
      end if
    until A new node is added to Q, or the heuristic value is lower than the bound
    b, or no assignment can resolve all the conflicts in C
  else
    if The solution set S is empty then
      Assume the objective value of the solution is v.
      Update the lower bound b ← v − c
    end if
    Add N to the solution set S
  end if
end while
Return the solution set S and the conflict set C.
```

Pareto front will be found. In the last section, I introduce the CDA* algorithm which is used in the bottom level. In this section, I introduce several multi-objective optimization methods that will be helpful when I introduce the top-level algorithm and the whole two-level algorithm in the next section.

As discussed in Section 1.1.1, making difficult decisions always involves addressing several objectives that are conflict with each other. Our goal is to help decision makers sift out the "good" solutions. But what kind of solutions are "good"? Let us take buying a computer as an example. An important criterion people always consider is the price. Another is often the performance and quality. A computer with higher price and lower quality than another could not be a "good" one. The good computers should not be both more expensive and more inferior, or in other words, a good computer should not be worse than another computer on every single criterion. Such intuition coincides with the definition of the Pareto front, which was introduced in Section 2.1.1. Here is a recap of the concept of Pareto front: Assume $\mathbf{g}$ represents the set of objective functions. A solution $\mathbf{x}$ is dominated by another solution $\mathbf{x}'$ if $\forall k,\ g_k(\mathbf{x}) \leq g_k(\mathbf{x}')$ (assume we are maximizing all g_i). In other words, a solution is dominated by another solution if none of its objective value is better than the other. A solution that is not dominated by any other solution is called an undominated solution, or Pareto optimal solution. All the undominated solutions form the complete Pareto front. A solution that is not dominated by any convex combination of other solutions is called an convex undominated solution, or convex Pareto optimal solutions. Formally, a solution $\mathbf{x}$ is convex Pareto optimal if and only if $\forall n \geq 1, \forall \lambda_{1..n}, \forall \mathbf{x}_{1..n},\ \exists\ k\ \ s.t.\ g_k(x_i) > \sum_{j=1}^{n} \lambda_j \cdot g_k(\mathbf{x}_j)$, where $\sum_{j=1}^{n} \lambda_j = 1,\ \lambda_j \geq 0$. All the convex undominated optimal solutions form the convex Pareto front. Figure 2-5 shows a two-dimensional Pareto front. Figure 2-6 shows the corresponding convex Pareto front. Point 1, for example, is on the Pareto front, but not on the convex Pareto front, since it is dominated by some points on the line connecting points 2 and 3.

The main goal of most multi-objective optimization methods is to reveal the complete Pareto front. Although the complete Pareto front would be an ideal result,

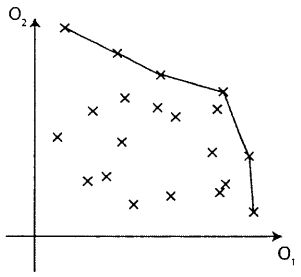


Figure 2-5: An example of Pareto front.

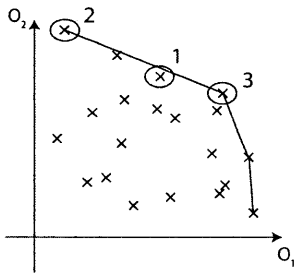


Figure 2-6: An example of convex Pareto front.

finding all the Pareto optimal solutions could be very time-consuming, especially when the number of solutions on the Pareto front is very large. It is sometimes more efficient to first approximate the shape of the complete Pareto front using the convex Pareto front and then focus on the area of interest identified by the user. For example, the convex Pareto front shown in Figure 2-6 has a similar shape as the complete Pareto front shown in Figure 2-6, yet only 4 points are on the convex Pareto front, compared to 6 points on the complete Pareto front. Potentially, we can save about 1/3 time by finding only the convex Pareto front.

This section introduces three multi-objective optimization methods. The first is the weighted sum approach, which can be used to find one solution on the convex Pareto front. The second is changing the objectives into constraints. The last are two close related algorithms: the Guided Improvement Algorithm (GIA) and the Opportunistic Improvement Algorithm (OIA).

2.3.1 Weighted Sum Approach

The weighted sum approach is probably the most straightforward idea for finding a Pareto optimal solution, given that a single-objective optimization tool is standing by: we put a weight on each objective and sum them into a single objective. For instance, if we have two objectives max $\{-A + 10B + 2C\}$ and max $\{7A - 5B - 2C\}$, and the weights are 0.4 and 0.6 respectively, the new objective is:

$$\max \{0.4 \times (-A + 10B + 2C) + 0.6 \times (7A - 5B - 2C) = 3.6A + B - 0.4C\}$$

More precisely, suppose there are n objective functions $g_1, g_2, \cdots g_k$, the problem can be expressed as follows:

$$\begin{aligned} \max \quad & \sum_{i=1}^{n} \lambda_i \, g_i \\ \text{s.t.} \quad & \sum_{i=1}^{n} \lambda_i = 1 \\ & \lambda_i \geq 0, \quad \forall i = 1, 2, \cdots n \end{aligned}$$

The optimal solution given by the weighted sum approach is guaranteed to be Pareto optimal [17]. Intuitively, if a solution $\mathbf{x}$ is dominated by another solution $\mathbf{x}'$, the weighted sum of the objective values of $\mathbf{x}$ could no be better than $\mathbf{x}'$, and hence cannot be the optimal solution of single-objective COP with the weighted objective. Not all Pareto optimal solutions, however, could be found by the weighted sum approach. Only the solutions that are on the CONVEX Pareto front could be found [18].

The next goal is to find all the solutions on the convex Pareto front using the weighted sum approach. A straightforward idea is to enumerate every possible set of weights. i.e., all the possible combination of $\lambda_1 \cdots lambda_n$. However, this idea is not realistic because the weights λ_i are continuous. Fortunately, using the recursive knee algorithm (introduced in Section 2.4.1), we can identify the sets of weights that are needed to find all the solutions on the convex Pareto front. I will discuss the

approach in detail in Section 2.4.

Before shifting to the next subsection, I would like to mention that the weighted sum approach is quite flexible for finding representative solutions that are evenly spread out on the front. By simply changing the weights, which could be viewed as the importance factor of each objective, the weighted sum approach can give decision makers the solution in different areas on the Pareto front.

2.3.2 Changing Objectives into Constraints

Although the weighted sum approach is straightforward, other ideas are needed to find the complete Pareto front. A widely used approach is to change objectives into constraints. For example, in a two-objective maximization problem, if we have a Pareto optimal solution whose objective values are (10, 20), to find another Pareto optimal solution, we can solve a new problem: $\max\{f_1\}$, s.t. $f_2 > 20$, or $\max\{f_2\}$, s.t. $f_1 > 10$. Generally, by solving the new problem with all but one objectives as constraints, we can find another Pareto optimal optimal solution. Figure 2-7 shows a more concrete example.

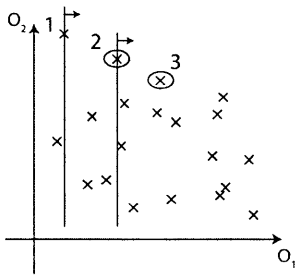


Figure 2-7: An illustration of how the "changing objectives into constraints" algorithm works.

In the example, we can solve the single objective problem that maximizes objective 2. Then we will have point 1 (assume the objective values are (v_1, v_2)). By adding the constraint $f_1 \geq v_1$ to the model, geometrically, we ask for the solution with maximum value of objective 2 among the ones that are on the right side of point 1, as indicated by the vertical line passing point 1 and the arrow in the graph. Then we solve the new problem and get point 2. Assume the objective values are (v_1', v_2'). Again, by

adding the constraint $f_1 \geq v_1'$ and solving the problem, we get the point with the maximum value of objective 2 on the right side of the line crossing point 2. Continue the procedure and finally we can get all the solutions on the Pareto front (Proved in [4]).

Essentially, the approach optimizes one objective, while setting thresholds for all the others. The most significant advantage of this approach is that it finds the complete Pareto, rather than the convex Pareto front only. However, there are also some shortcomings. The most undesirable one is that if the objective function is in a complicated form, for instance, involving nonlinear or functions like logarithm, it is hard to be encoded and used as a normal constraint. Of course we can always check the consistency when we have a full assignment, but we lose the edge of using conflict learning and other constraint processing techniques like forward and backward checking. Moreover, the approach essentially uses one objective to iteratively block an area in which the solutions are eliminated and limits the search in the unblocked areas. We can combine several objectives to block a larger area to reduce the search space. The new approach is discussed in the next subsection.

2.3.3 Guided Improvement Algorithm (GIA) and Opportunistic Improvement Algorithm (OIA)

In this subsection, I introduce two similar algorithms that solve multi-objective COP by repeatedly transforming multi-objective COP into Constraint Satisfaction Problems (CSP). The two algorithms are Guided Improvement Algorithm (GIA) and Opportunistic Improvement Algorithm (OIA) [30].

The basic idea of GIA is that it finds a Pareto optimal solution (refer to Section 2.1.1 for definitions) by solving a series of CSPs at each stage. After each stage, the area that is dominated by the newly found Pareto optimal point is forbidden in the subsequent search by adding constraints to the CSPs. The process of GIA is shown in Figure 2-8. The corresponding pseudo codes are shown in Algorithm 4 and Algorithm 5.

Algorithm 4 The Guided Improvement Algorithm (GIA)

Input: A COP = (CSP, **g**)
Returns: All solutions on the complete Pareto front

Solution set $S \leftarrow \{\}$
repeat
  Call Algorithm 5 to find a Pareto optimal solution
  if A solution s_{new} is returned **then**
    Update solution set $S \leftarrow S + s_{new}$
    Assume the solution corresponds to point $p = (p_1, p_2, \cdots, p_n)$
    Add a constraint $((g_1 \geq p_1)$ OR $(g_2 \geq p_2)$ OR $\cdots$ OR $(g_n \geq p_n))$ into the CSP constraint set
  end if
until No more solution can be found
Return the solution set S

Algorithm 5 Find one Pareto optimal solution (subroutine of Algorithm 4)

Input: A COP = (CSP, **g**)
Returns: A Pareto optimal solution

A working CSP' $\leftarrow$ CSP
Solution $s \leftarrow$ *null*
repeat
  Call a CSP solver to solve the working CSP'
  if A solution is returned **then**
    $s \leftarrow$ solution returned by the CSP solver
    Assume solution s corresponds to point $p = (p_1, p_2, \cdots, p_n)$
    Add constraints $(g_1 \geq p_1)$ AND $(g_2 \geq p_2)$ AND $\cdots$ AND $(g_n \geq p_n)$ into the constraint set of CSP'
  end if
until No more solution can be found
Return solution s

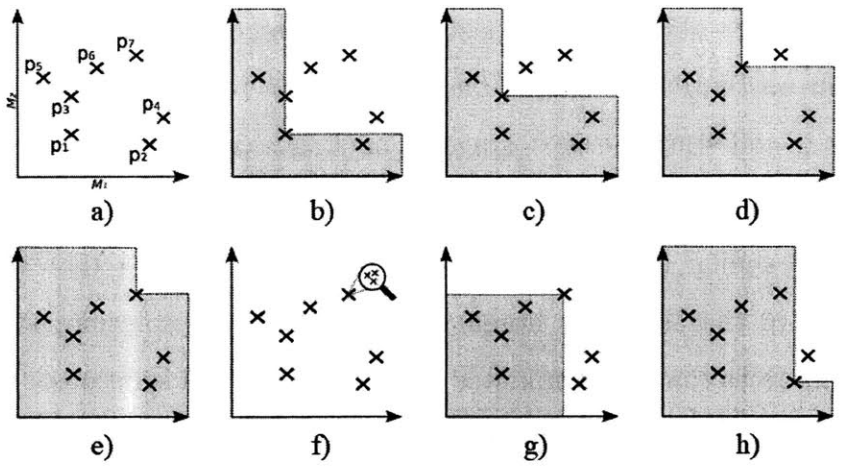


Figure 2-8: An illustration of the Guided Improvement Algorithm (GIA).

Recall that COP can be expressed as COP = (CSP, $\mathbf{g}$), where $g_i \in \mathbf{g}$, $i = 1, \cdots, n$ are n objective functions (refer to Section 2.1.1 for definitions). At the beginning of the first stage, GIA ignores the objective functions and solves the CSP only. After finding a solution $\mathbf{x_1}$, n constraints are temporarily added to the CSP, restricting the search space in the area in which the points dominate the point corresponding to $\mathbf{x_1}$. For example, in Figure 2-8, point p_1 is found at the first step. Assume its objective values are $(g_1, g_2) = (10, 12)$, then two constraints $g_1 \geq 10$ and $g_2 \geq 12$ are temporarily added to the constraint set. The effect is shown in sub-figure b): the solutions in the shaded area are no longer feasible. In the next step, the algorithm solves a new CSP with the n extra constraints again. If it finds a new solution, add two new constraints into the constraint set again and continue the process until no more solution can be found. Sub-figures a) to e) in Figure 2-8 show such a process. When no more feasible solution can be found as shown in sub-figure f), it means no other solution dominates the last found solution, or in other words, we find a Pareto optimal solution. Now remove all the temporary constraints that are added to the constraint set in the previous steps. Add a permanent constraint which reduces the search space by eliminating the area that is dominated by the Pareto optimal solution we just found. For example, if the last found solution corresponds to $(g_1, g_2) = (40, 50)$, add a permanent constraint: $(g_1 > 40)$ OR $(g_2 > 50)$ into the constraint set. We call the process stated above as a stage. In one stage, GIA repeatedly generates CSPs which

translate the result obtained from last CSP as temporary constraints, and finally reach a point on the Pareto front. Then GIA add a permanent constraint into the constraint set, and start the next stage. The stages continue until no more Pareto optimal solution can be found, when all the solutions on the Pareto front are collected.

OIA uses a similar idea as the one used in GIA, but rather than restricting the feasible space in the area that dominates the last found solution, it continuously eliminates the areas that are dominated by the newly found solutions. The basic idea is shown in Figure 2-9. The pseudo code is shown in Algorithm 6.

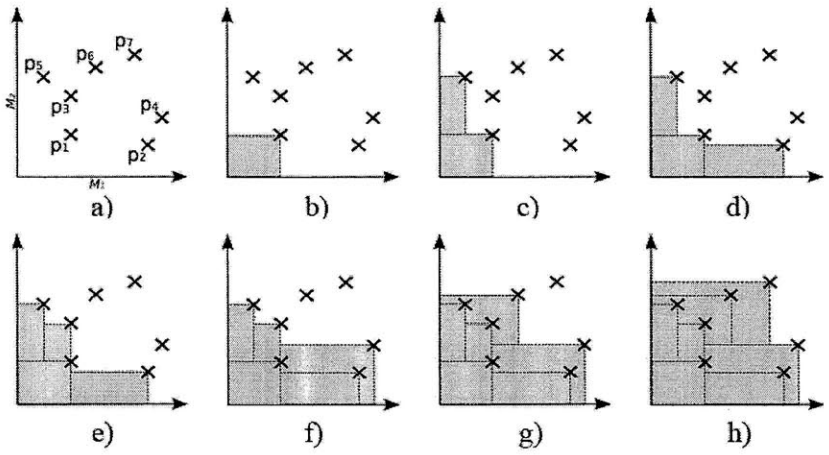


Figure 2-9: An illustration of the Opportunistic Improvement Algorithm (OIA)

Algorithm 6 The Opportunistic Improvement Algorithm (OIA)

Input: A COP = (CSP, **g**)
Returns: All solutions on the complete Pareto front

Solution set $S \leftarrow \{\}$
repeat
  Call a solver to solve the CSP
  if A solution s_{new} is returned **then**
    Update solution set $S \leftarrow S + s_{new}$
    Assume the solution corresponds to point $p = (p_1, p_2, \cdots, p_n)$
    Add a constraint $((g_1 \geq p_1)$ OR $(g_2 \geq p_2)$ OR $\cdots$ OR $(g_n \geq p_n))$ into the CSP constraint set
  end if
until No more solution can be found
Return the solution set S

At the first step, OIA solves the CSP problem without considering any objective function. A solution is found (point 1). Assume the solution corresponds to point $(p_1, p_2, \cdots, p_n)$, a constraint $((g_1 \geq p_1)$ OR $(g_2 \geq p_2)$ OR $\cdots$ OR $(g_n \geq p_n))$ is added to the constraint set of the CSP. Geometrically, the new constraint reduces the search area by eliminating the shaded area shown in sub-figure b). The algorithm repeats such a process, and adds more constraints, as shown in sub-figures c) to g), until no more solution can be found in the remaining search space. It means no other solution dominates the current set of solutions, i.e., all the undominated solutions are found, as we can see in sub-figure h).

GIA and OIA have two significant advantages. First, they obtain the complete Pareto front, rather than the convex Pareto front only. Second, they eliminate an area in search space after solving each CSP. It helps to accelerate the search in the subsequent steps. These good features are complementary to the ones of the weighted sum approach and the algorithm that changes objectives into constraints. In Section 2.4.3, I propose an algorithm that improves my algorithm by mixing it with OIA.

In this section, I introduce three methods that are helpful to solve multi-objective COPs. The first approach I introduce is the weighted sum approach. It gives each objective a weight and generate a new objective that is a sum of the weighted objective functions. The approach can be used to find one solution on the convex Pareto front. The second method is changing objective functions into constraints. It optimizes only one objective function and sets a threshold for all the other objective functions. The last algorithms are GIA and OIA. They transform the multi-objective COPs into a series of CSPs which lead to the complete Pareto front. The ideas of the three methods will be used in the next section, where I develop my algorithm that solves multi-objective COPs.

2.4 Solving Multi-objective COPs

This section introduces several novel algorithms for solving multi-objective COPs. As mentioned in the last two sections, I use a two-level algorithm to solve the multi-

objective COP: The top level generates a single objective based on the objective function set repeatedly, and the bottom level takes the single objective and uses CDA* (introduced in Section 2.2) to solve the COP. The series of single objectives generated by the top level guarantees that all the solutions on the convex Pareto front will be found.

The first algorithm I develop in this section is called "recursive knee algorithm". The recursive knee algorithm is an algorithm that is used to approximate continuous convex Pareto front[32]. It has never been used to solve multi-objective COPs which have discrete solutions. I introduce the algorithm into the discrete domain and develop the recursive knee algorithm that works with CDA* to solve multi-objective COPs. In general, the discrete recursive knee algorithm generates a series of single-objectives that are weighted sums of the original objective functions. By solving the COPs with these single-objective functions using CDA*, all the solutions on the convex Pareto front can be found. After introducing the discrete recursive knee algorithm, I extend it to find all solutions in a region close to the convex Pareto front. The shape and size of the region can be controlled by a set of parameters. Finally, I introduce an algorithm that combines the discrete recursive knee algorithm and the Opportunistic Improvement Algorithm (OIA) introduced in Section 2.3.3. The mixed algorithm can find all solutions on the complete Pareto front, instead of only the convex Pareto front.

2.4.1 The Discrete Recursive Knee Algorithm

This subsection introduces the discrete recursive knee algorithm and shows how to use the algorithm to solve multi-objective COPs. I first explain the concept of a "knee" solution, and then briefly introduce the idea of the recursive knee algorithm. After providing the pseudo code, I elaborate the algorithm with an example of multi-objective COP.

A "knee" solution on a continues Pareto front is defined by I. Das in [12]. Figure 2-10 shows a knee point on a continuous two-objective Pareto front (maximizing both objectives). As shown in the figure, at the knee point, a small improvement in one

objective will cause a large deterioration in the other, which might make decision makers reluctant to move in either direction.

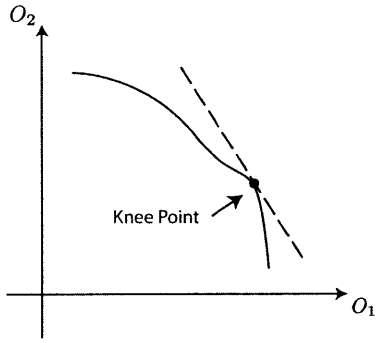


Figure 2-10: An example of the knee point.

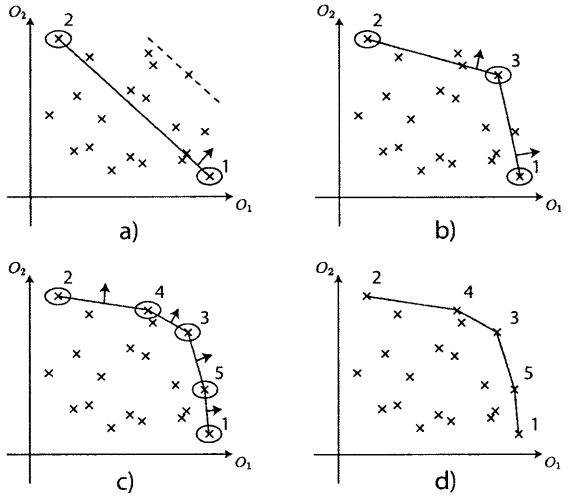


Figure 2-11: A geometric illustration of the discrete recursive knee algorithm.

I develop an algorithm that combines the recursive knee algorithm and CDA* to solve multi-objective COPs. Figure 2-11 illustrates the algorithm when solving a two-objective COP. In this figure, both objectives are maximized. The discrete recursive knee algorithm adopts a recursive procedure that repeatedly finds knee solutions on the convex Pareto front. As shown in Figure 2-11, the algorithm starts with optimizing each individual objective in the objective function set. More precisely, for the multi-objective COP $(\mathbf{x}, \mathbf{D_x}, C_x, \mathbf{g})$ (refer to Section 2.1.1 for the detailed definition of the multi-objective COP), optimize the single-objective COPs $(\mathbf{x}, \mathbf{D_x}, C_x, g_i)$, for every $g_i \in \mathbf{g}$. In the two-dimensional example, the algorithm takes each of the two

objectives g_1, g_2 and optimize each of them as a single-objective COP with the original variables, domains and constraints. In Figure 2-11, points 1 and 2 correspond to the optimal solutions for objective 1 and objective 2, respectively. These two points are called base solutions because the next single objective function is calculated based on the two points.

The next step is to calculate the weight vector $\lambda = (\lambda_1, \cdots, \lambda_n)$. The new single-objective function will be $g = \sum_{i=1}^{n} \lambda_i \cdot g_i$. In order to calculate the weight vector λ, we need to calculate the direction vector λ' of the line connecting the two base solutions in the graph first. The weight vector λ represents the direction that is orthogonal to direction λ'. For example, assume that point 1 is $(g_1, g_2) = (35, 10)$ and point 2 is $(g_1', g_2') = (5, 25)$. The direction vector of the line connecting the two points is $\lambda' = (g_1 - g_1', g_2 - g_2') = (35 - 5, 10 - 25) = (30, -15)$. Then we have $\lambda = (1, 2)$, which is orthogonal to λ' since $lambda' \cdot lambda^T = 0$. Use $\lambda_1, \cdots, \lambda_n$ as the weights for each of the objectives, and we can get a new single objective that is a weighted sum of all the objective functions. In the example, the new objective function is $g = \lambda \cdot \mathbf{g} = g_1 + 2g_2$.

Generally, for n-objective COPs, there are n base solutions. The n base solutions form a $n-1$ dimensional hyperplane and the weight vector represents the orthogonal direction of this hyperplane. For example, Figure 2-12 shows a 3-objective case. The points b_1, b_2 and b_3 are three base points. The arrow indicates the direction that is orthogonal to the 2-dimensional plane of b_1, b_2, b_3. Generally, the weight vector λ is orthogonal to any line that connects two base solutions like the lines b_1b_2, b_1b_3 and b_2b_3. More precisely, assume the n base solutions are $b_1 = (b_{11}, b_{22}, \cdots, b_{1n})^T$, $b_2 = (b_{21}, b_{22}, \cdots, b_{2n})^T$, $\cdots$, $o_n = (o_{n1}, o_{n2}, \cdots, o_{nn})^T$, respectively. Since weight vector λ should be orthogonal to the line connecting any two base solutions, the inner product of λ and $b_i - b_j$ $(\forall\ i \neq j)$ should be 0. Let matrix $A = \begin{bmatrix} b_1 - b_2 & b_1 - b_3 & \cdots & b_1 - b_n \end{bmatrix}$. We have $A^T \cdot \lambda = 0$, i.e., the weight vector λ is the null vector of matrix A. Hence, simply using Gaussian elimination, we can determine the weight vector λ based on the matrix A.

Back to Figure 2-11, geometrically, maximizing this new objective is equivalent to

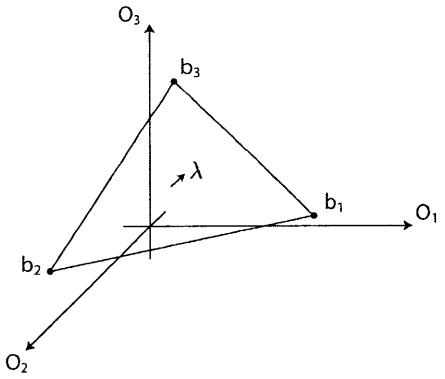


Figure 2-12: A geometric illustration of a 3-objective weight vector. λ represents the weight vector, which is orthogonal to the plane connecting the three base solutions b_1, b_2 and b_3.

pushing the line connecting points 1 and 2 in the direction that is orthogonal to the line (indicated by the arrow) until the farest point is reached. In the example, point 3 is the farest point in this direction. By definition, this is the knee point. The graph is then divided into two intervals by the knee point 3. Points 1 and 3 form a new set of base solutions, so do points 2 and 3. The procedure is then recursively executed on both intervals. It stops when the knee solution is no better than the base solutions when the new objective is optimized. It geometrically means no points that is farer in the direction that is orthogonal to the line connecting the two base solutions can be found. For example, when we try to push the line connecting points 2 and 4 in the direction indicated by the arrow, we can find no more points in that direction.

In sum, the discrete recursive knee algorithm repeatedly calculates a set of weights according to the direction that is orthogonal to the line connecting the base solutions. Using the weighted sum approach introduced in Section 2.3.1, the algorithm generates a new objective that is a weighted sum of all the objective functions and use CDA* to get the optimal solution to the new function. The pseudo code of the discrete recursive knee algorithm is shown in Algorithm 7 and Algorithm 8.

The discrete recursive knee algorithm has pros and cons. One of the pros is that, analytically, it could be run efficiently because the recursive steps only rely on the number of solutions on the convex Pareto front, rather than the solution space. We shall see this in the time complexity analysis (Section 2.5.1), using multi-objective

Algorithm 7 Find convex Pareto optimal solutions of a multi-objective COP using the discrete recursive knee algorithm

Input: A multi-objective COP = (CSP, $\mathbf{g}$), where CSP = $(\mathbf{x}, \mathbf{D_x}, C_x)$, and the objective function set $\mathbf{g} = (g_1, \cdots, g_n)$. Refer to Section 2.1.1 for definitions.
Returns: The solutions on the convex Pareto front

Solution set $S \leftarrow \{\}$
Base solutions $B \leftarrow \{\}$
Conflict set $C \leftarrow \{\}$. This is a global variable.
for $i = 1$ to n **do**
  Use CDA* (Algorithm 2) to solve the single-objective COP = (CSP, g_i) with initial conflict set C.
  Assume the solution s_{new} and the conflict set C_{new} are returned.
  Add solution s_{new} into solution set S
  Add solution s_{new} into base solution set B
  Update conflict set $C \leftarrow C_{new}$
end for
Call Algorithm 8 with the multi-objective COP and base solution set B. C is used as a global variable.
Assume solution set S' is returned
Update solution set $S \leftarrow S + S'$
Return solution set S

Algorithm 8 The recursive process of finding knee solutions.

Input: A multi-objective COP = (CSP, $\mathbf{g}$); Base solution set $B = (b_1, \cdots, b_n)$
Returns: Solution set S

Solution set $S \leftarrow \{\}$
Calculate the weights $\lambda = (\lambda_1, \cdots, \lambda_n)$ using the objective values of base solutions
New objective function $g \leftarrow \sum_{i=1}^{n} \lambda_i \cdot g_i$
Call CDA* (Algorithm 2) to solve the single-objective COP = (CSP, g) with the initial conflict set C (C is a global variable).
Assume solution s_{new} (the knee solution) and conflict set C_{new} are returned
Update conflict set $C \leftarrow C_{new}$
Update solution set $S \leftarrow S + s_{new}$
if $g(s_{new}) > g(b_i)$ **then**
  for i = 1 to n **do**
    New base solution set $B' \leftarrow B - b_i + s_{new}$
    Call Algorithm 8 recursively with the COP and the new base solution set B'
    Add the returned solutions into solution set S
  end for
end if
Return solution set S

A* as a comparison. Another pro is that we can limit the depth of the recursive procedure and still get a satisfactory result. I. Das studied the properties of the knee points and pointed out that several knee points on the continuous Pareto front would be enough to approximate the shape of the convex Pareto front [11]. It means that if there is a large amount of convex Pareto optimal solutions, we can limit the depth to shortens the running time yet we can still approximate the shape of the convex Pareto front effectively. The downside of the algorithm is that it cannot find all solutions on the complete Pareto front, but only the convex Pareto front. It is because that we use the weighted sum approach which is unable to find the solutions that are Pareto optimal but not CONVEX Pareto optimal. An improved approach that can recover the complete Pareto front is proposed in Section 2.4.3.

To better demonstrate the whole process of using the discrete recursive knee algorithm with CDA* to solve multi-objective COPs, I will illustrate every step of solving the two-objective COP shown in Figure 2-13. There are 3 decision variables with 3, 2, and 4 options respectively. A single constraint limits the difference between B and C to be no less than 2. There are two objectives that are conflict with each other: objective 1 wants to minimize A, while maximizing B and C, and objective 2 wants to maximize A and minimize B and C. To provide a global view of the search space, I enumerate all feasible solutions and plot the convex Pareto front in Figure 2-14.

- Decision variables:

  1. Decision $A = \{2, 4, 7\}$
  2. Decision $B = \{3, 8\}$
  3. Decision $C = \{2, 5, 6, 9\}$

- Constraint:

$$|B - C| \geq 2$$

- Objectives:

  1. max $\quad -A + 10B + 2C$
  2. max $\quad 7A - 5B - 2C$

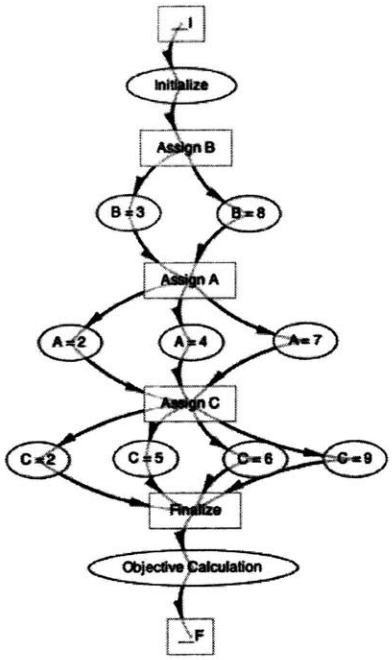


Figure 2-13: An example of two-objective COP, used for demonstration of the discrete recursive knee algorithm.

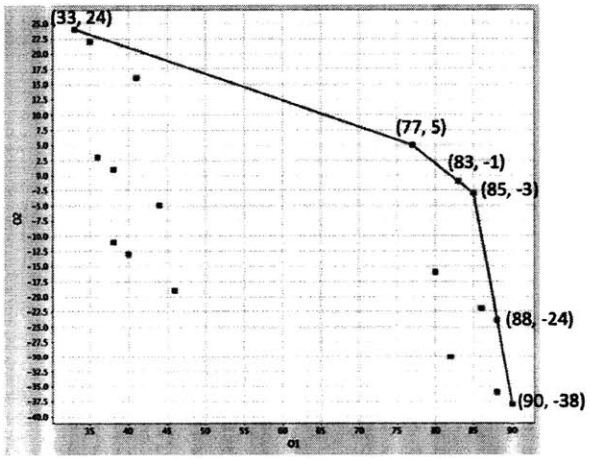


Figure 2-14: All solutions of the two-objective COP shown in Figure 2-13

First, the algorithm optimizes Objective 1: max $-A + 10B + 2C$. The search tree is shown in Figure 2-15. We have already seen the process in Section 2.2.1, but it is reviewed again as a recap. According to the dynamic ordering rule used in CDA*, the algorithm expands the variable with the smallest domain size first. Notice that the objective function has the MPI property. The $B = 8$ branch is expanded

first and Node 2 is obtained. Similarly, the algorithm expands $A = 2$ branch and gets Node 3. Next, the branch $C = 9$ is expanded and a leaf Node 4 is reached. However, the constraint $|B - C| = 1 < 2$ is violated because of the partial assignment $(B = 8, C = 9)$. Therefore, the conflict ($B = 8$ and $C = 9$) is recorded. Next, the algorithm expands Nodes 1, 2 and 3 again and gets Nodes 5, 6 and 7, with their heuristic values shown next to the nodes. Now we choose Node 6 (the one with the largest heuristic value) to expand. The best branch without the known conflicts is $C = 6$, but this node has a smaller heuristic value than node 7. Notice that we can still get a better solution by expanding Node 2 again. Thus, Node 2 is expanded and Node 9 is obtained. The heuristic value of Node 9 is 91, which is higher than the heuristic value of Node 7. But when expanding Node 9, the algorithm gets Node 10 with the heuristic value smaller than Node 7. Thus, Node 7 is examined and it is consistent. Hence, Node 7 is the best solution for this objective function. The corresponding objective value pair is (90, -38).

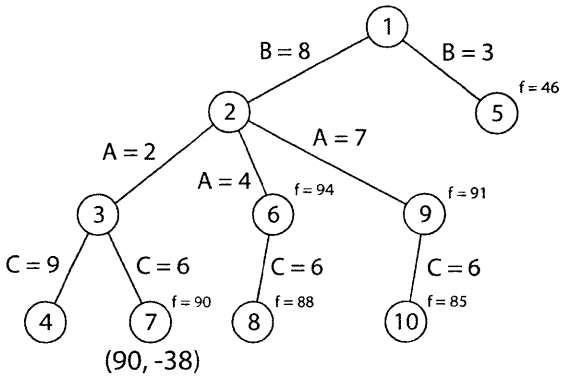


Figure 2-15: Search tree of objective 1

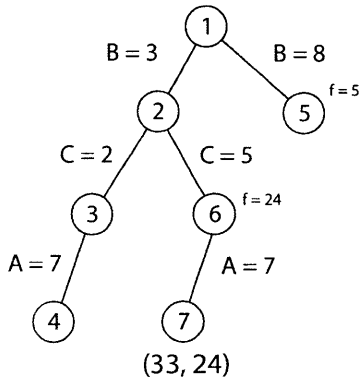


Figure 2-16: Search tree of objective 2

Similarly, Figure 2-16 shows the search tree of objective 2: max $7A - 5B - 2C$. Note that the sequence of assigning decision variables has been changed. The difference is caused by the conflict that we got from the first optimization step. Since $B = 9$ and $C = 8$ is a known conflict, the algorithm assigns B and C at the beginning to avoid this conflict. When the leaf Node 4 is reached, the algorithm finds it inconsistent because of violating the constraint $|B - C| \geq 2$. Thus, only Nodes 1 and 2 are expanded. By calculating the heuristic values, the algorithm chooses to expand Node

6 and obtains the best feasible solution, Node 7. The corresponding objective value pair is (33, 24). Also, the algorithm captures another conflict ($B = 3$ and $C = 2$).

Now we need to calculate the weight for each objective. In the example, the objective values of point 1 is $(g_1, g_2) = (90, -38)$, and let the objective values of point 2 is $(g_1', g_2') = (33, 24)$. The direction vector of the line connecting the two points is $(g_1 - g_1', g_2 - g_2') = (57, -62)$. Then the vector of the direction that is orthogonal to the $(57, -62)$ is $(62, 57)$, or $(0.52, 0.48)$ if normalized. That is, we have $(\lambda_1, \lambda_2) = (0.52, 0.48)$. This is the weight vector for the new objective. Numerically, the new objective is: $g = \lambda_1 g_1 + \lambda_2 g_2 = 0.52 \times (-A + 10B + 2C) + 0.48 \times (7A - 5B - 2C) = 2.83A + 2.81B + 0.84C$. This is the objective 3.

Thanks to the conflicts that we got from the first two phases, the search is now more efficient. Figure 2-17 shows the expanding tree of the new objective. Only three nodes are expanded before the optimal solution is reached. The conflict directed search becomes smarter because of the information it preserved in the previous steps. This learning process helps the single objective solver to stay away from inconsistent solutions in the future searches. The effect shall become more and more obvious as the search continues.

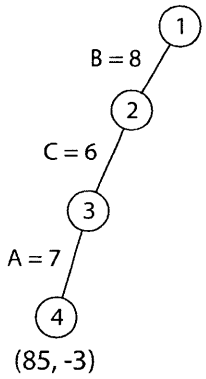


Figure 2-17: Search tree of objective 3

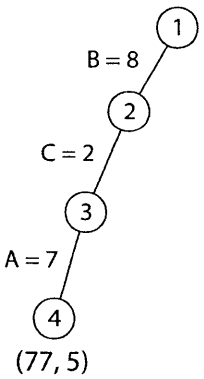


Figure 2-18: Search tree of objective 4

Figure 2-18 presents the search tree of objective 4, which uses solutions of objective 2 and objective 3 as base solutions. The weights for the objectives are 0.34 and 0.66 respectively. Figure 2-19 shows the solutions we have got until now.

A new phenomenon shows up in the next step. Geometrically, one might have

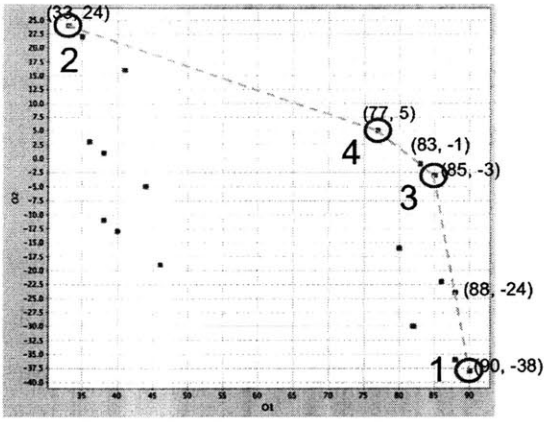


Figure 2-19: The four Pareto optimal points that are found until step 4.

noticed that a solution (88, -24) lies on the line connecting the solution for objective 1 (80, -38) and the solution for objective 3 (85, -3). Solving the problem with objective 4, therefore, should bring us this new point (88, -24) as well as the two base solutions. This phenomenon also reflects on objective 4: $0A + 8.125B + 1.5C$. $0A$ means as long as the solution does not violate the constraint, A could be assigned any value without changing the objective value. In Figure 2-20, we can see that three nodes turn out to be optimal, which is in line with the geometric and algebraic analyses. Since we also get the two base solutions at this step, we know there is no more convex Pareto optimal solutions can be found within this interval. Thus, we stop the recursion within this interval.

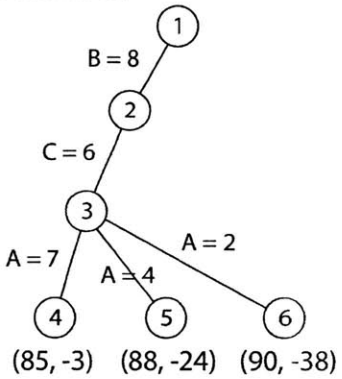


Figure 2-20: Search tree of objective 5

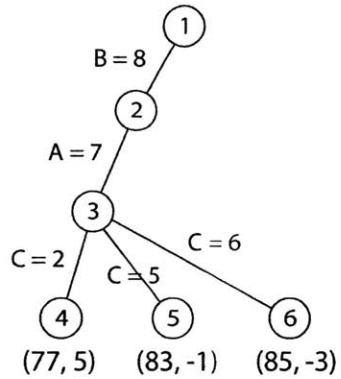


Figure 2-21: Search tree of objective 6

Similarly, when it comes to the interval between points (77, 5) and (85, -3), the weights are (0.5, 0.5) and the objective function is $3A + 2.5B + 0C$. In this case,

decision variable C could be assigned any value without influencing the objective value. The expanding tree is shown in Figure 2-21. We find another solution (83, -1) along with the two base solutions. With the base solutions being found, we are also done with this interval.

Finally, we have to search within the interval between points (33, 24) and (77, 5). The coefficients are (0.3, 0.7) and the last objective is $4.6A - 0.48B - 0.79C$. The expanding tree is shown in Figure 2-22. We get nothing new but the base solutions. The search in this interval stops. There are no more intervals to be explored now. After 7 steps, we establish the convex Pareto front.

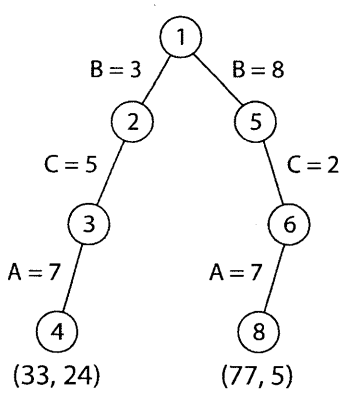


Figure 2-22: Search tree of objective 7

In sum, I develop the discrete version of recursive knee algorithm in this subsection. The algorithm works with CDA* to find all the solutions on the convex Pareto front. The next subsection extends the algorithm so that it finds all solutions in a region that are close to the convex Pareto front. The size and shape of the region can be adjusted by a set of parameters.

Moreover, a shortcoming of the discrete recursive knee algorithm is that the algorithm is unable to find all the solutions on the Pareto front. That is, the Pareto optimal solutions hiding in the convex hull is unreachable. This shortcoming originates from the limitation of the weighted sum approach introduced in Section 2.3.1. I will propose an improved algorithm that can find the complete Pareto front in Section 2.4.3.

2.4.2 Finding Solutions in an Area Close to the Convex Pareto Front

This subsection extends the discrete recursive knee algorithm introduced in the last subsection to obtain not only the solutions exactly on the convex Pareto front, but also the ones that are in an area that is close to the convex Pareto front. The size and shape of the area can be adjusted by a set of parameters.

In many decision making scenarios, the decision maker needs not only the solutions on the Pareto front, but also the ones near the Pareto front. There are two important reasons. The first concern is the possible inaccuracy of the data. Either rounding error or random sampling could cause bias in the data. It implies that the solutions on the Pareto front might not actually be as good as some other ones that are close enough to the front if the bias dominates the distance between the solutions and the Pareto front. Furthermore, consider a multi-phase decision-making problem in which the data are is computed approximately or estimated in the first phase, and the relatively good solutions are then extracted and reconsidered in the following phases. In this case, getting solutions that are close to the Pareto front is almost as important as getting solutions that are exactly on the Pareto front under such circumstances. On the other hand, when the data is not numerical, it also makes sense to consider solutions close to the front. For instance, when one needs to evaluate the importance of different factors that could impact a project, an expert might be invited to give a score to each factor. The scores are not precise at all and are very likely to be an over or under estimate. Hence, a small score difference might not correctly reflect a preference over different solutions. Sometimes it is important to have those solutions under considerations as well.

The other concern is the robustness of the solutions. For example, when designing a spaceship, one architecture might be Pareto optimal for exploring the lunar surface, but is awful for a Mars mission, while another architecture, although it is not Pareto optimal, is very close to the Pareto front in both missions. The latter architecture is preferred most of the time, since it saves a large amount of effort that would be

spent on designing and manufacturing different types of components for two different systems. Under this circumstance, a decision maker might want to compare the Pareto front regions and pick common solutions, rather than focusing on the Pareto front only.

Now I show how to extend the discrete recursive knee algorithm to find solutions close to the Pareto front. First I introduce the notations that are needed to describe the algorithm. Assume there are n objective functions, then there are n base solutions at each step. Further assume that the objective values of the i'th base solution are $b_{i1}, b_{i2}, \cdots, b_{in}$, $i = 1, \cdots, n$, that is, b_{ij} represents the j'th objective value of the i'th base solution. Point $\mathbf{b_i} = (b_{i1}, b_{i2}, \cdots, b_{in})^T$ is called the i'th base point. Let the weight vector for the set of base solutions be $\mathbf{w} = (w_1, w_2, \cdots, w_n)$, where w_i is the weight for the i'th objective. Then the new objective $g = \mathbf{w} \cdot \mathbf{g} = \sum_{i=1}^{n} w_i \cdot g_i$. Note that the new objective value for the i'th base solution is $\mathbf{w} \cdot \mathbf{b_i}$. Also, since the weight vector is perpendicular to the hyperplane determined by all the base points, we have $\mathbf{w} \cdot (\mathbf{b_i} - \mathbf{b_j}) = 0, \forall i \neq j$, or equivalently, $\mathbf{w} \cdot \mathbf{b_i} = \mathbf{w} \cdot \mathbf{b_j}$, which means the new objective values of all the base solutions are the same. Denote this value v_b.

To find the solutions in an area close to the Pareto front, we need decision maker to set a set of cutoff values $\mathbf{c} = (c_1, c_2, \cdots, c_n)^T$, where c_i sets the tolerance level for objective i when objective i is being optimized without considering any other objectives. For example, if the optimal value of objective 1 is 90 and $c_1 = 10$, when optimizing objective 1, all the solutions whose objective values are greater than 90 - 10 = 80 will be returned. The tolerance level for the new objective is $c_{new} = \mathbf{w} \cdot \mathbf{c}$. In other words, all the solutions with the new objective value greater than $v_b - c_{new}$ will be returned.

Recall that in addition to the optimal solution, the CDA* algorithm can also find the second best solution, third best solution, etc., until a certain number of solutions are found or a tolerance level is reached. Combining this feature of CDA* and the discrete recursive knee algorithm, we can extend the recursive knee algorithm to achieve our goal. The basic idea is shown in Figure 2-23. Generally, when a recursive step finds no better solution than the base solutions, we ask CDA* not to stop but

to return all the solutions within a tolerance level. Geometrically, we pull back the line connecting the base solutions in the direction indicated by the arrows shown in sub-figure b) of Figure 2-23 and collect all the solutions between the pulled-back line and the original line. These solutions are circled in Figure 2-23. The shape and size of the area between the convex Pareto front and pulled back lines can be controlled by the cutoff value set $\mathbf{c} = (c_1, c_2, \cdots, c_n)^T$. To be more specific, if the cutoff values are large, the area will be large, while if they are small, the area will be slim. Also, the shape can be controlled by adjusting the relations between the cutoff values. The pseudo code of the extended algorithm is shown in Algorithm 9 and Algorithm 10.

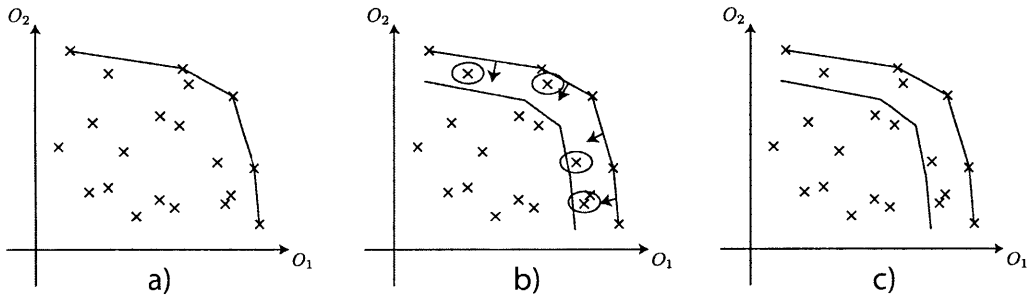


Figure 2-23: A geometric illustration of how to find solutions in an area close to the Pareto front.

Decision Variables	A, B, C
Options	A = {2, 4, 7} B = {3, 8} C = {2, 5, 6, 9}
Constraint	$\|B - C\| \geq 2$
Objective	max $-A + 10B + 2C$ max $7A - 5B - 2C$

Table 2.5: A two-objective COP, used for illustration of the algorithm that finds solutions close to the convex Pareto front.

To better illustrate the idea, I use the two-objective COP shown in Figure 2-13 again. The COP is shown in Figure 2.5 again for reader convenience. Recall that when we search in the interval between points (85, -3) and (90, -38) (objective 5, Figure 2-20), the weight vector is (0.875, 0.125), the objective function is $0A+8.125B+1.5C$, and we stop when we find the three optimal solutions (85, -3), (88, -24) and (90, -38).

Algorithm 9 Find the solutions in an area close to the convex Pareto front using the discrete recursive knee algorithm

Input: A multi-objective COP = (CSP, **g**); A cutoff value set $\mathbf{c} = (c_1, c_2, \cdots, c_n)$
Returns: The solutions on the convex Pareto front

Solution set $S \leftarrow \{\}$
Base solutions $B \leftarrow \{\}$
Conflict set $C \leftarrow \{\}$. This is a global variable.
for $i = 1$ to n **do**
  Use generalized CDA* (Algorithm 3) to solve the single-objective COP = (CSP, g_i) with initial conflict set C and cutoff value c_i.
  Assume solution set S_{new} and conflict set C_{new} are returned.
  Update solution set $S = S + S_{new}$
  Add the best solution in S_{new} into the base solution set B
  Update conflict set $C \leftarrow C_{new}$
end for
Call Algorithm 10 with the multi-objective COP, base solution set B and cutoff value set **c**. C is used as a global variable.
Assume solution set S' is returned
Update solution set $S \leftarrow S + S'$
Return solutions set S

Assume the cutoff for objective 1 is 8, and the cutoff for objective 2 is 16, then the cutoff value for objective 5 should be $8 \times 0.875 + 16 \times 0.125 = 9$. Thus, we keep searching until the objective value decreases by 9. Figure 2-24 shows the extended search tree.

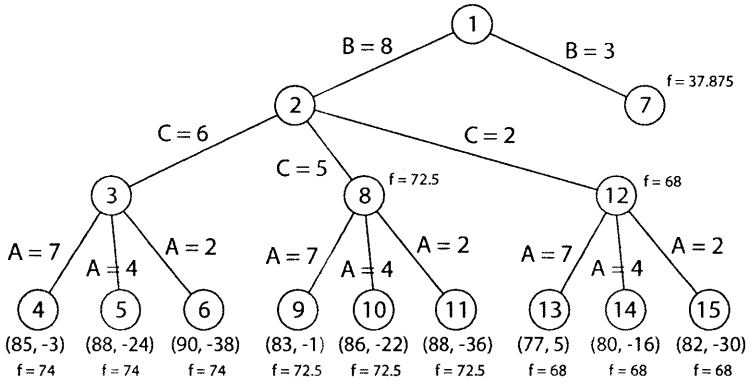


Figure 2-24: Extended search tree of objective 5

In Figure 2-20, we can see that the search stops when it expands node 6. Since we have a cutoff value 9, the algorithm continues the search until the objective value

Algorithm 10 The extended recursive process of finding solutions close to the convex Pareto front

Input: A multi-objective COP = (CSP, **g**); Base solution set $B = (b_1, \cdots, b_n)$; Cutoff value set $\mathbf{c} = (c_1, \cdots, c_n)$
Returns: Solution set S

Solution set $S \leftarrow \{\}$
Calculate the weights $\lambda = (\lambda_1, \cdots, \lambda_n)$ using the objective values of base solutions
New objective function $g \leftarrow \sum_{i=1}^{n} \lambda_i \cdot g_i$
Call CDA* (Algorithm 2) to solve the single-objective COP = (CSP, g) with initial conflict set C.
Assume solution s_{new} (the knee solution) and conflict set C_{new} are returned
Update conflict set $C \leftarrow C_{new}$
Update solution set $S \leftarrow S + s_{new}$
if $g(s_{new}) > g(b_i)$ **then**
  for i = 1 to n **do**
    New base solution set $B' \leftarrow B - b_i + s_{new}$
    Call Algorithm 10 recursively with the COP, the new base solution set B' and the cutoff value set **c**
    Add the returned solutions into solution set S
  end for
else
  Calculate the cutoff value $c_v \leftarrow \sum_{i=1}^{n} \lambda_i \cdot c_i$
  Call generalized CDA* (Algorithm 3) to solve the single-objective COP = (CSP, g) with initial conflict set C and cutoff value c_v
  Assume solution set S_{new} and conflict set C_{new} are returned
  Update conflict set $C \leftarrow C_{new}$
  Update solution set $S \leftarrow S + S_{new}$
end if
Return solution set S

drops to 74 - 9 = 65. Now nodes 1 and 2 are expanded again. The heuristic value of node 7 is 37.875, which is smaller than 65. Hence, we discard this node. On the other hand, the heuristic value of node 8 is 72.5, which is greater than 68, thus, node 8 is expanded. When it comes to node 12, the heuristic value equals 68, which is still greater than 65, so node 12 is also expanded. At this stage, we stop as we have exhausted all consistent solutions whose objective value is greater than 65. Four new solutions, (86, -22), (88, -36), (80, 16) and (82, -30) are the ones that are not convex Pareto optimal, but are close enough to the Pareto front. The points are circled at the bottom-right corner in Figure 2-25. When the same search is performed on objectives 6 and 7, all the points circled in Figure 2-25 are found. Those are the points that are close to the Pareto front.

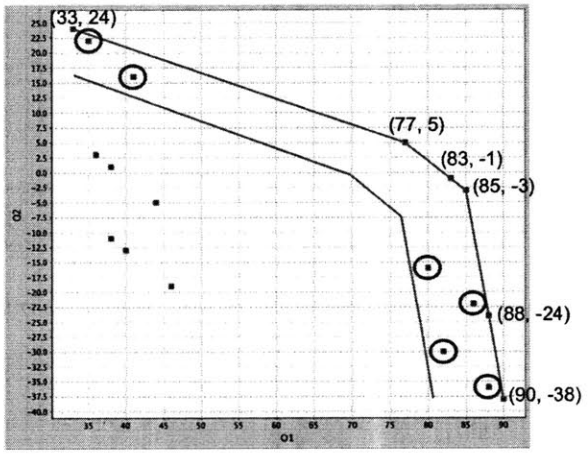


Figure 2-25: Finding solutions in an area close to the Pareto front. The solutions on and between the two blue lines are the solutions found by Algorithm 9. The circled solutions are the ones that are not on the convex Pareto front but are close to the convex Pareto front. The shape and size of the area between the two blue lines can be controlled by adjusting the cutoff values of each objective.

The cutoff vector provides substantial flexibility for decision makers. For instance, if one of the objectives is obtained from precise computation, like weight or cost, while the other is a subjective estimate, like importance or risk, the decision maker could simply set a tight cutoff for the precise objective value and a loose bound for the estimated one. By adjusting the tolerance level for different objectives, the decision makers can restrict the search in an area of interest and reduce the number of returned

solutions.

In this subsection, I extend the discrete recursive knee algorithm introduced in the last subsection. The extended algorithm can help decision makers to identify the solutions that are not exactly on the convex Pareto front, but are close enough to the convex Pareto front. In the next subsection, I propose an algorithm that combines the discrete recursive knee algorithm and the Opportunistic Improvement Algorithm (OIA) introduced in Section 2.3.3. The new algorithm can find the solutions on the COMPLETE Pareto front, rather than the solutions on the convex Pareto front only.

2.4.3 A Mixed Algorithm Combining the Discrete Recursive Knee Algorithm and the Opportunistic Improvement Algorithm (OIA)

I develop a mixed algorithm combining the discrete recursive knee algorithm introduced in Section 2.4.1 and the Opportunistic Improvement Algorithm (OIA) introduced in Section 2.3.3. The main goal of developing this algorithm is to overcome the shortcoming of the discrete recursive knee algorithm which cannot find all solutions on the COMPLETE Pareto front, but only the CONVEX Pareto front.

As mentioned in Section 2.3.3, OIA repeatedly generates new constraints that eliminate an area that is dominated by a point in the search space. It uses the objective functions as constraints, but does not use them to guide the search. Its advantage is that it finds the complete Pareto front, rather than the convex Pareto front only. The recursive knee algorithm on top of the CDA* algorithm, on the other hand, fully exploits the information that is implied by the objective functions and use it to guide the search all the way to the Pareto optimal solutions. However, it could only reach the convex Pareto front. As analyzed above, the advantages of the two algorithms are complementary. It motivates us to mix the two algorithms to develop a new approach that use the objective functions to guide the search and finds all solutions on the complete Pareto front. Figure 2-26 demonstrates the basic idea of the mixed algorithm. Algorithm 11 and Algorithm 12 provide the corresponding

pseudo code.

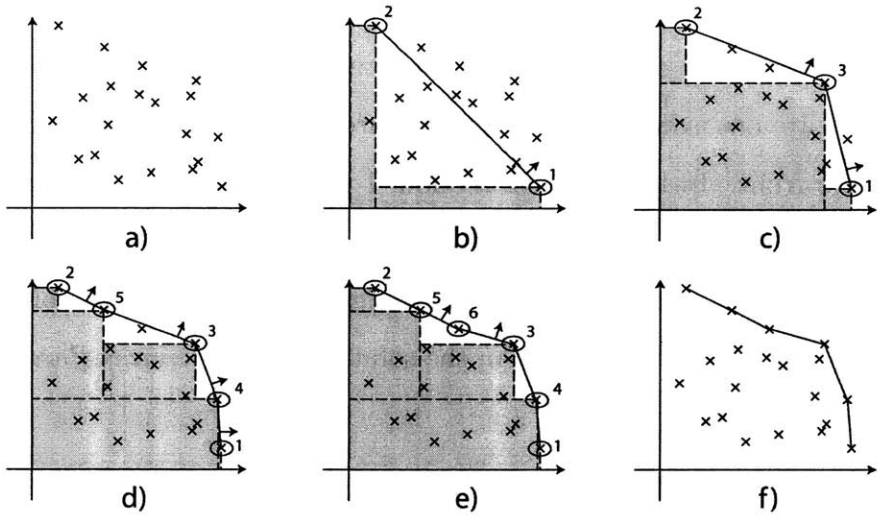


Figure 2-26: A geometric illustration of the improved discrete recursive knee algorithm. It combines the discrete recursive knee algorithm with the Opportunistic Improvement Algorithm (OIA) which eliminates the area dominated by the newly found solution by adding a new constraint to the multi-objective COP.

Similar to the discrete recursive knee algorithm, the algorithm starts with optimizing each objective. Points 1 and 2 are found, as shown in sub-figure a). Assume point 1 and 2 correspond to $\mathbf{b_1} = (b_{11}, b_{12})$ and $\mathbf{b_2} = (b_{21}, b_{22})$, respectively, then the algorithm adds two constraints $((g_1 > b_{11})$ OR $(g_2 > b_{12}))$ and $((g_1 > b_{21})$ OR $(g_2 > b_{22}))$ to the constraint set. Geometrically, the two constraints shade the region that is dominated by points 1 and 2. Since the points in the shaded region could not be Pareto optimal (they are dominated by points 1 and 2), the algorithm will not search these areas in the future. Next, the algorithm calculates a weight for every objective using the approach described in Section 2.4.1. Using the new weighted sum of objective functions to search the unshaded region, the algorithm reaches point 3, as shown in sub-figure c). Again, the algorithm shades the region that is dominated by point 3 by adding a constraint $((g_1 > b_{31})$ OR $(g_2 > b_{32}))$ to the CSP, assuming point 3 is $b_3 = (b_{31}, b_{32})$.

The process is then executed recursively on both the interval between points 1 and 3 and the one between points 2 and 3. As a result, point 4 and 5 are found in the unshaded zone, as shown in sub-figure d). Note that point 5 is NOT CONVEX

Algorithm 11 Find solutions on the COMPLETE Pareto front combining the discrete recursive knee algorithm and OIA

Input: A multi-objective COP = (CSP, **g**)
Returns: The solutions on the COMPLETE Pareto front

Solution set $S \leftarrow \{\}$
Base solutions $B \leftarrow \{\}$
Conflict set $C \leftarrow \{\}$. This is a global variable.
for $i = 1$ to n **do**
  Use CDA* (Algorithm 2) to solve the single-objective COP = (CSP, g_i) with initial conflict set C.
  Assume the solution s_{new} and the conflict set C_{new} are returned. Further assume the objective values of solution s_{new} are $(b_1, b_2, \cdots, b_n)$
  Add solution s_{new} into solution set S
  Add solution s_{new} into base solution set B
  Update conflict set $C \leftarrow C_{new}$
  Add a constraint $((g_1 > p_1)$ OR $(g_2 > p_2)$ OR $\cdots$ OR $(g_n > p_n))$ into the CSP constraint set
end for
Call Algorithm 12 with the multi-objective COP and base solution set B. C is used as a global variable.
Assume solution set S' is returned
Update solution set $S \leftarrow S + S'$
Return solution set S

Algorithm 12 The recursive process of finding knee solutions on the COMPLETE Pareto front

Input: A multi-objective COP = (CSP, **g**); Base solution set $B = (b_1, \cdots, b_n)$
Returns: Solution set S

Solution set $S \leftarrow \{\}$
Calculate the weights $\lambda = (\lambda_1, \cdots, \lambda_n)$ using the objective values of base solutions
New objective function $g \leftarrow \sum_{i=1}^{n} \lambda_i \cdot g_i$
Call CDA* (Algorithm 2) to solve the single-objective COP = (CSP, g) with the initial conflict set C (C is a global variable).
if A solution is found **then**
  Assume the solution s_{new} and the conflict set C_{new} are returned. Further assume the objective values of solution s_{new} are $(b_1, b_2, \cdots, b_n)$
  Update conflict set $C \leftarrow C_{new}$
  Update solution set $S \leftarrow S + s_{new}$
  Add a constraint $((g_1 > p_1)$ OR $(g_2 > p_2)$ OR $\cdots$ OR $(g_n > p_n))$ into the CSP constraint set
  for i = 1 to n **do**
    New base solution set $B' \leftarrow B - b_i + s_{new}$
    Call Algorithm 12 recursively with the COP and the new base solution set B'
    Add the returned solutions into solution set S
  end for
end if
Return solution set S

Pareto optimal, but only Pareto optimal. Geometrically, the point is on the inner side of the line connecting points 2 and 3. If no constraint were added to eliminate the dominated area (the shaded area), points 2 and 3 would have a better objective value than point 5. However, because points 2 and 3 are in the areas that are eliminated by the algorithm in the previous steps, point 5 becomes the best. In other words, points 2 and 3 become inconsistent because of the constraints added in the previous steps.

The same situation occurs when the algorithm searches in the interval between points 3 and 5, as shown in sub-figure e). Point 6, which is not convex Pareto optimal, but only Pareto optimal, is found. This is the last step. There is no more solution found in all the intervals, thus the procedure terminates. The final result is shown in sub-figure f). Note that the algorithm successfully obtains the solutions on the COMPLETE Pareto front.

This subsection introduces a mixed algorithm combining the discrete recursive knee algorithm and OIA. The mixed algorithm uses the objective function to guide the search, and finds all the solutions on the COMPLETE Pareto front. Till now, I have introduced all the three algorithms I develop to solve multi-objective COPs. The algorithms take a multi-objective COP which is defined in Section 2.1.1, and solve it combining the CDA* algorithm introduced in Section 2.2 and several multi-objective optimization methods introduced in Section 2.3. In the next section, I analyze the time and space complexity of the discrete recursive knee algorithm which works with CDA*.

2.5 Complexity Analysis for Multi-objective CDA*

In this section, I analyze the time and space complexity of the combination of the discrete recursive knee algorithm and CDA*. I also analyze the complexities of the mixed algorithm described in Section 2.4.3. As a comparison, I analyze the time and space complexity of the multi-objective A* algorithm [40] as well.

2.5.1 Time Complexity Analysis

I analyze the time complexity in this subsection. Let us consider the discrete recursive knee algorithm first. I need some notations for the analysis before I start:

n – The number of decision variables.

m – The maximum number of options for the decision variables.

k – The number of objective functions.

p – The number of solutions on the convex Pareto front.

Since the recursive knee algorithm calls a single-objective solver at each step, we need to first analyze the time complexity of single-objective CDA* and then the maximum possible number of steps. In the worst case, at each step, CDA* has to visit every possible assignment of decision variables, that is, it has to expand the whole search tree. The worst time complexity for each step is therefore $O(m^n)$.

Next, let us consider the number of steps. At each step, when the recursive knee algorithm calls the single objective solver, one of two possible results will be returned: either at least one new solution is returned, or the base solutions are returned. For the former case, since there are at most p solutions on the convex Pareto front, the number of total steps is bounded by p. For the latter case, because every solution on the convex Pareto front could only be a base solution for k times, the total number of such steps should not exceed kp. For example, when there are two objectives, a convex Pareto optimal solution could at most be a base solution twice: for the interval on its left and the one on its right. Thus, the total number of steps is bounded by $(k+1)p$, or $O(kp)$.

In sum, the worst case time complexity for the recursive knee algorithm is $O(kpm^n)$. It means that the time complexity of the recursive knee algorithm is linear of the number of objectives and the number of solutions on the convex Pareto front, but is exponential of the number of decision variables and the maximum number of options for the decision variables.

The time complexity analysis of the mixed algorithm is the same as above, except the notation p does not mean the number of the solutions on the convex Pareto front, but the number of solutions on the complete Pareto front.

As a comparison, the worst case time analysis for the multi-objective A* is provided here. In the worst case, which happens when the last node expanded in the tree dominates all other solutions, all possible combinations of decision variables are enumerated. Thus, in the last level, there are $O(m^n)$ nodes. To determine whether a node is dominated by another, all the objective values have to be compared, which costs $O(k)$ time. According to the algorithm, a node has to be compared to all other leaf nodes to test Pareto optimality. Thus, it takes $O(km^n)$ time to test the Pareto optimality for one node. Further, since all m^n nodes have to be checked, the total time complexity is $O(m^n \cdot km^n) = O(km^{2n})$. It is linear of the number of objectives, but exponential of the number of decision variables and the square of the maximum possible options.

Let us briefly compare the two algorithms. The recursive knee algorithm takes time $O(kpm^n)$ and the multi-objective A* takes time $O(km^n m^n)$. The only difference is that the second factor is p for the recursive knee algorithm while it is m^n for the multi-objective A*. Notice that m^n is the total number of solutions in the entire search space, while p is just the number of solutions on the convex Pareto front. In almost all problems in practice, $p << m^n$. Thus, the recursive knee algorithm outperforms the multi-objective A* algorithm in the time complexity analysis.

2.5.2 Space Complexity Analysis

The space complexity of the discrete recursive knee algorithm is easy to analyze. Since it calls the single objective solver in every step and not keeping the search tree, the space complexity of the algorithm is the same as the space complexity of the single-objective CDA*. In the worst case, the tree will be fully expanded, which takes $O(m^n)$ space. The space complexity of the mixed algorithm is exactly the same as the recursive knee algorithm. Notice that it is irrelevant to the number of objective functions.

As a comparison, the multi-objective A* takes $O(m^n)$ space to store tree nodes in the worst case. However, it has to keep all the objective values in each node, which makes its space complexity $O(km^n)$ in total.

	Recursive knee	Multi-objective A*
Time complexity	$O(kpm^n)$	$O(km^n m^n)$
Space complexity	$O(m^n)$	$O(km^n)$

Table 2.6: The comparison of time and space complexity between the multi-objective A* algorithm and the discrete recursive knee algorithm.

As shown in Table 2.6, the discrete recursive knee algorithm and the mixed algorithm introduced in Section 2.4 outperform the multi-objective A* algorithm both in time complexity and space complexity analysis. The analysis also shows that the recursive knee algorithm is a relatively good algorithm since it linearly depends on the number of solutions on the convex Pareto front. Intuitively, no matter what an algorithm does, it always has to reach all the convex Pareto solutions, which should at least take a linear time of p.

This subsection provides the time and space complexity analysis for the algorithms I develop in Section 2.4. The results show that my algorithm is theoretically advanced and might outperform the multi-objective A* algorithm when solving multi-objective COPs. I summarize this chapter in the next section.

2.6 Summary

To solve real-world decision problems, a two-step scheme is adopted in this chapter. The first is to model them with multi-objective COPs, and the second is to solve the multi-objective COPs. Correspondingly, this chapter first introduces the definition of multi-objective COP. The similarity of the structure of multi-objective COP and real-world decision-making problems motivates me to use multi-objective COP to model decision-making problems. I also provide a detailed introduction of how to model each part of decision-making problems as multi-objective COPs.

In the subsequent sections, I discuss how to solve multi-objective COPs. I use a two-level algorithm to solve multi-objective COPs: The top level generates a single objective based on the objective function set repeatedly, and the bottom level takes the single objective and uses CDA* to solve the COP. The series of single objectives

generated by the top level guarantees that all the solutions on the convex Pareto front will be found. Corresponding to the two levels, I first introduce the CDA* algorithm and its generalization in Section 2.2, and then introduce three multi-objective optimization methods in Section 2.3. Finally, I develop three algorithms combining CDA* and multi-objective optimization methods to find solutions on the CONVEX Pareto front, in an area close to the convex Pareto front, and on the COMPLETE Pareto front, respectively.

As shown in the Section 2.5, the algorithms I described in this chapter are efficient and flexible. With the help of the algorithms, decision-makers can construct a rough model first and then find the common features of the good solutions. Finally, they can further polish the model according to the information they exploit from the rough model to obtain the best solution.

In the next chapter, I provide two case studies of real-world problems. I use the approach introduced in Section 2.1.2 to model the two problems. The results supports the claim that we can model real-world decision-making problems as multi-objective COPs.

Chapter 3

Applications of Optimal Decision Making

In the last chapter, I elaborate the similarity between the structures of the multi-objective Constraint Optimization Problem (COP) and the decision-making problems. They both consist of four elementary parts: a set of decision variables, a finite domain for each decision variable, a set of constraints defining the relations between the decision variables, and a set of objective functions which are conflict with each other in some degree. Such resemblance motivates me to develop a series of methods and algorithms to model decision-making problems with multi-objective COPs and to solve the multi-objective COPs. I first introduced how to model decision-making problems with multi-objective COPs, and then illustrate how to combine various single-objective COP-solving algorithms and multi-objective optimization methods to find the Pareto front of multi-objective COPs.

To show how to use multi-objective COP to encode practical problems, and to show the effectiveness of the algorithms I develop, I introduce two real-world decision-making problems and study how to use COP to model and solve them in this chapter. The first problem is the study of the Apollo lunar exploration[36]. The complex system structure and the difficulty of choosing the mission mode makes it a good example of real-world decision-making problem. The second problem is the study of the NASA decadal survey[9][35]. The decadal survey aims for identifying the sequence

of launching a series of satellites which are used for earth observation. It represents a class of practical decision-making problems which requires to provide an optimal order of a set of missions. I elaborate how to model the two problems in the COP framework, and show the results obtained from the COP solver which is developed based on the algorithms introduced in Chapter 2.

3.1 Apollo Architecture Study

The Apollo lunar exploration program in 1960s is arguably the most ambitious and unprecedented engineering challenges in human history. The reason to choose this case is that even in today's perspective, it is still a difficult problem. The most critical part of the project is to make the decision of the mission mode. Two years were spent on debating the choice of mission mode before the final decision is made on June, 1962. The goal of the debate is to chose an optimal mission mode by considering launch complexity and total mission risk. After the stage of choosing the mission mode, the whole project rapidly proceed with the detailed design and development of the spacecraft.

This section shows how the decision-making problem of the Apollo mission mode fits into the COP framework, which consists the four elementary parts: decision variables, a finite domain for each decision variable, constraints, and objective functions. Correspondingly, I demonstrate the decision variables and their domains first, then the constraints between the decision variables, and finally the objectives.

Based on the principle that the most critical decisions of the mission mode should be made first, a set of nine decision variables were selected for the study [36]. The set of decision variables includes the ones related to the mission mode, the crew size, and the rocket fuel types used for Apollo. Table 3.1 shows the decision variables and the options for each of them. Figure 3-1 is an illustration of the different mission modes.

During the Apollo architecting process that took place in 1960s, three different classes of misson modes were under consideration: direct, earth orbit rendezvous, and lunar orbit rendezvous. The explanation of three mission modes in four of the

Decision	Abbreviation	Option 1	Option 2	Option 3	Option 4
Earth Orbit Rendezvous	EOR	No	Yes		
Earth Launch Type	EL	Orbit	Direct		
Lunar Orbit Rendezvous	LOR	No	Yes		
Arrival at Moon	AM	Orbit	Direct		
Departure from Moon	DM	Orbit	Direct		
Command Module Crew	CMC	2	3		
Lunar Module Crew	LMC	0	1	2	3
Service Module Fuel	SMF	Cryogenic	Storable		
Lunar Module Fuel	LMF	N/A	Cryogenic	Storable	

Table 3.1: A set of nine key decisions for the Apollo study.

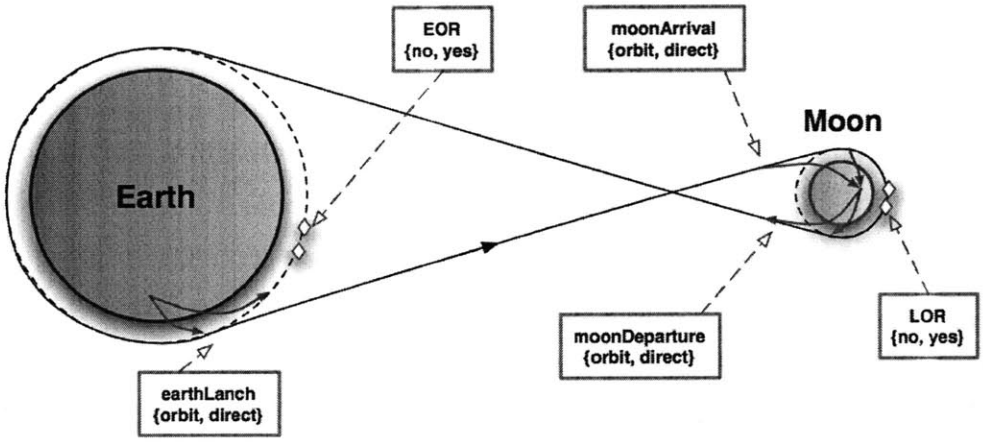


Figure 3-1: The illustration of Apollo mission modes [36]. If EOR is yes, it means to rendezvous two modules in the earth orbit, as indicated by the blue arrow pointing to the earth orbit. If LOR is yes, it means to rendezvous two modules in the lunar orbit, as indicated by the blue arrow pointing to the lunar orbit.

decisions of Table 3.1 is shown in Table 3.2. For example, for the direct mission mode, a spacecraft would travel directly to the moon, landing and returning as a unit. This plan would have required a very powerful booster, the planned Nova rocket. For the EOR mission mode, multiple rockets would be launched, each carrying various parts of a direct ascent spacecraft and propulsion units that would have enabled the spacecraft to escape earth orbit. After a docking in earth orbit, the spacecraft would have landed on the moon as a unit. For the LOR mission mode, one Saturn V would launch a spacecraft that was composed of modular parts. A command module would remain in orbit around the moon, while a lunar module would descend to the moon and then return to dock with the command module while still in lunar orbit. In

contrast with the other plans, LOR required only a small part of the spacecraft to land on the Moon, thereby minimizing the mass to be launched from the moon's surface for the return trip. Some further discussion and explanation of the decision variables can be found in [36].

Apollo mission mode	Description
Direct mission mode	EOR is no, EL is orbit or direct, LOR is no, and MD is orbit or direct.
EOR mission mode	EOR is yes, EL is orbit, MA is orbit or direct, LOR is no or yes, and MD is orbit or direct.
LOR mission mode	EOR is no or yes, EL is orbit or direct, MA is orbit, LOR is yes, and MD is orbit.

Table 3.2: Historical Apollo mission modes under consideration

The constraints are generated by capturing available knowledge about the system and the relationships between the decision variables. Table 3.3 shows the constraint list, followed by a brief explanation of each constraint. The further discussion of each constraint can be found in [36].

Name	Scope	Constraint
EOR constraint	EOR, EL	(EOR = no) or (EOR = yes and EL = orbit)
LOR constraint	LOR, MA	(LOR = no) or (LOR = yes and MA = orbit)
Moon leaving	LOR, MD	(LOR = no) or (LOR = yes and MD = orbit)
Crew size	CMC, LMC	CMC $\geq$ LMC
Lunar module crew	LOR, LMC	(LOR = no and LMC = 0) or (LOR = yes and LMC > 0)
Fuel constraint	LOR, LMF	(LOR = no and LMF = n/a) or (LOR = yes and LMF $\neq$ n/a)

Table 3.3: List of constraints between decision variables [36]

- EOR constraint: If there is an earth orbit rendezvous, the earth launch decision must be equal to orbit, since it is impossible to rendezvous without entering the earth orbit first.

- LOR constraint: If there is a lunar orbit rendezvous in the mission mode, the moon arrival decision must be equal to orbit, since it is impossible to complete the rendezvous maneuver without entering lunar orbit before descending to the lunar surface.

- Moon departure: If there is a lunar orbit rendezvous in the mission mode, the moon departure decision must be equal to orbit, since it is impossible to complete the rendezvous maneuver without entering lunar orbit after ascending from the lunar surface.

- Crew size: This constraint restricts the crew size of the lunar module to be less than or equal to the crew size of the command module.

- Lunar module crew: This constraint forces lunar module crew size to be zero if there is no lunar orbit rendezvous.

- Fuel constraint: This constraint forces the fuel type of the lunar module to be n/a if there is no lunar orbit rendezvous.

There are many objectives that are needed to be optimized when choosing the mission mode. Among the most important metrics, the operational risk and the Initial Mass To Low Earth Orbit (IMLEO) are the two that is especially useful to predict the success of the Apollo project.

Following common aerospace engineering practice, the rocket equation can be used to estimate vehicle masses, depending on the velocity increment they must supply[6]. The equation can be defined as:

$$m_f = m_i \exp\left(\frac{-\Delta V}{g_0 \cdot I_{sp}}\right)$$

where ΔV is the difference in velocity over the entire period of the maneuver, g_0 is the gravitational constant, I_{sp} is the specific impulse of the propulsion system, m_f is the final mass after the maneuver, and m_i is the initial mass before the maneuver. The two mass terms m_f and m_i can be broken down as $m_f = m_{bo} + m_{pl}, m_i = m_{bo} + m_{pl} + m_{prop}$, where m_{bo} is the burnout mass, m_{pl} is the payload mass, and m_{prop} is the propellant mass. For a multi-stage rocket system, the rocket equation can be applied recursively for each maneuver. If the payload of a stage is actually another rocket with its own fuel and payload, then m_{pl} becomes the initial mass for the next application of the equation. In this study, values for constants such as the

structural mass ratios, propulsion characteristics, and models for crew compartment sizes are taken from a combination of historic data and the assumptions used in the contemporary 1961 Houbolt Report [25]. When designing the mission mode, we want to minimize the IMLEO metric, so that the launch vehicle from the earth surface can be of "reasonable" size.

Decision	Option 1	Option 2	Option 3	Option 4
Earth Orbit Rendezvous	No	Yes		
	0.98	0.95		
Earth Launch Type	Orbit	Direct		
	0.99	0.90		
Lunar Orbit Rendezvous	No	Yes		
	1	0.95		
Arrival at Moon	Orbit	Direct		
	0.99	0.95		
Departure from Moon	Orbit	Direct		
	0.90	0.90		
Command Module Crew	2	3		
	1	1		
Lunar Module Crew	0	1	2	3
	1	0.90	1	1
Service Module Fuel	Cryogenic	Storable		
	0.95^{burns}	1		
Lunar Module Fuel	N/A	Cryogenic	Storable	
	1	0.95^2	1	

Table 3.4: List of probability of success of each decision

Other than IMLEO, the operational risk is another very important metric for the mission success. For the computational reason, I adopt the complementary metric "probability of success" here. A mission mode's probability of success is one minus the operational risk. The probability of success of the whole mission is the product of the probability of success of each mode choice. For example, the probability of success of EOR is 0.95 (1 means successful, 0 means no chance of success), and the probability of LOR is also 0.95. If EOR and LOR were both adopted, the total probability of success would reduce to $0.95 \times 0.95 = 0.9025$. The values of probability of success are shown in Table 3.4. The risk of using cryogenic fuel in the service module depends on the number of burns performed by the module's engine. The number of burns

depends on the LOR decision variable: if LOR is no, the engine of the service module must fire four times, while if LOR is yes, only twice[36]. When making decisions for the mission mode, we want to maximize the probability of success.

So far, I have introduced all the four elements of the Apollo study that are needed in a COP: decision variables, a finite domain for each decision variable, constraints and objectives. The complete COP model is shown in Appendix A.

The results are shown in Figure 3-2, 3-3, and 3-4. In all the three plots, the X axis represents the probability of success and the Y axis is the IMLEO metric. Figure 3-2 shows all the feasible mission modes. It is obtained by Willard Simmons[36] (Figure 4-8 in [36]). The convex Pareto front is portrayed by the blue line connecting all the convex undominated points.

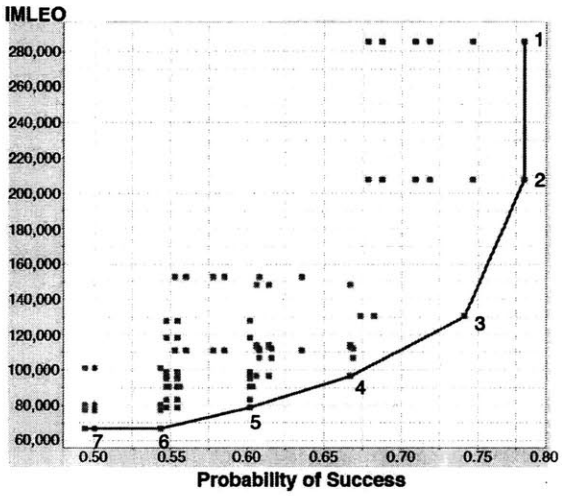


Figure 3-2: All feasible mission modes. The blue line shows the convex Pareto front. Point 3 corresponds to the Apollo mission mode that is finally adopted.

Figure 3-3 plots the results obtained by the tool designed base on the discrete recursive knee algorithm and CDA* that I introduced in Chapter 2. Comparing Figure 3-2 and Figure 3-3, we can see that my algorithm correctly finds all the solutions on the convex Pareto front. Points 1 to 7 in Figure 3-2 matches the points 1 to 7 in Figure 3-3. Points 1 and 2 correspond to the direct mission mode with 3 crew members and 2 crew members, respectively. As mentioned above, these mission modes send the spacecraft as a whole from earth directly to the moon. Neither EOR nor LOR is used. The advantage of these modes is that they avoid the rendezvous, and thus

reduce the overall mission risk. However, the two modes require a spacecraft with a high IMLEO. It increases the difficulty of designing and transporting the whole spacecraft. Points 3 to 7 are architectures which include LOR. As we can see in Figure 3-3, Point 3 is a "knee point" on the convex Pareto front (refer to Section 2.4.1 for detailed explanation of knee point). More specifically, at a knee point, a small improvement in one objective causes a relatively large deterioration in the other. Thus, a knee point is usually of great interest to the decision makers. In the Apollo project, Point 3 is finally chosen as the mission mode for Apollo. It has three crew members in the command module, two crew members in the lunar module and uses storable propellants for both the service module and the lunar module. Point 7 has the minimum mass, but is the most risky. It uses two crew members in a command module and one crew member in a lander with cryogenic propellants. Point 7 is the point which most closely models the proposed Soviet lunar mission's architecture.

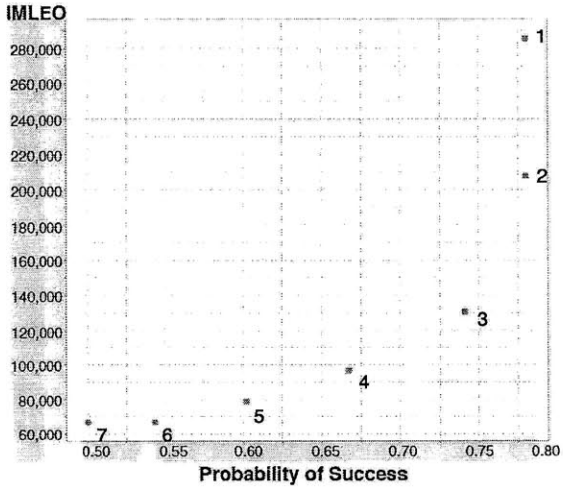


Figure 3-3: The convex Pareto front of the Apollo mission mode study, obtained by solving the multi-objective COP model.

Figure 3-4 shows all the solutions near the convex Pareto front. The cutoff value for IMLEO is 20,000, and the cutoff value for the probability of success is 0.05. Geometrically, the cutoff values of the two objectives determine the shape and size of the area between the two blue lines. The IMLEO cutoff value determines the gap between the blue lines in the bottom-left corner, and the cutoff value of the probability of success determines the width of the blue lines in the top-right corner.

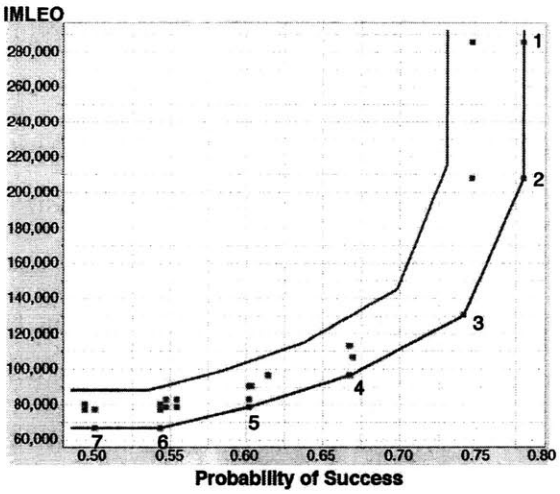


Figure 3-4: Solutions in an area close to the convex Pareto front. All the solutions in the region between the two blue lines are found. The bottom-left gap and the top-right width of the blue lines are determined by the cutoff parameters.

Refer to Section 2.4.2 for the rigorous definition and detailed explanation of the cutoff values. Figure 3-4 shows us the potentially good mission modes. It also explains why Point 3 is finally chosen: no other points are close to Point 3, which means when the area near Point 3 is of interest, the only choice for the decision makers is Point 3.

This section introduces the decision-making case study for the choice of Apollo's mission mode. I elaborate how to use multi-objective COP to model the decision variables, domains, constraints and objective functions of the Apollo mission mode. The results obtained by solving the multi-objective COP model using the algorithms I develop in Chapter 2 are shown and discussed. They are consistent with the results obtained in [36], but are obtained in much less time. In the next section, I introduce another case study, the earth science decadal survey.

3.2 Earth Science Decadal Survey Study

This section introduces another case study, the earth science decadal survey study. In order to coordinate the earth science space observation missions, the National Research Council commissioned a decadal survey report in 2004 for earth science to establish a unified agenda for the next several decades[35]. The long-term agenda

would promote cost-sharing opportunities and eliminate redundancies while ensuring the benefits of the earth science research.

The report chose 17 important measurement sets that are of significant value for the study of earth science. Corresponding to the 17 measurement sets, there are as many as 26 satellites (also called "missions" throughout this section) needed to be launched. The goal of the decadal survey is to decide on the relative priority and urgency of the measurement sets but not the detailed architecture of the missions to make the measurements. The decadal survey proposed a united set of 17 missions that they postulated would fit in the NASA budget. However, subsequent study have indicated that some of the proposed missions might better be devoided of, all are more expensive than projected and the budget more constrainted. The costs of building and launching of some of the satellites that are likely to be the earlier ones are shown in Table 3.5. The goal of this analysis is to find an optimal sequence of these missions, within a constraint of a budget of 300 million dollars per year. Another factor under consideration in the choice of mission sequence is the TRL (Technology Readiness Level) date. The TRL date of a mission (a satellite launch) indicates the earliest possible date that the mission can be launched. The TRL dates of the possible early missions are shown in Table 3.5.

In addition to the budget and TRL constraints, two important objectives need to be optimized: the total benefit derived from the sequence of the missions and the fairness. Six major research communities get benefits from the missions:

- Human Health and Security (HHS)
- Land-use, Ecosystems and Bio-diversity (LEB)
- Solid Earth hazards, natural resources and dynamics (SE)
- Climate Variability and Change (CVC)
- Weather Science and Applications (WSA)
- Water Resources and the global hydrological cycle (WR)

For each community, there is a delegating panel in NASA. The benefits for each panels of launching the satellites that are more likely to be the early ones are shown in Table 3.5. The benefits, however, depreciate along the years. That is, the later a satellite is launched, the smaller benefit each panel receives. The benefit depreciation rate for each panel is also shown in Table 3.5. Another objective is the fairness for the panels. As we can see in Table 3.5, a mission is beneficial to some of the panels, but are not helpful at all to some other panels. That is, the benefit received by each panel is different when a satellite is launched. When determining the sequence of the missions, a certain degree of fairness has to be achieved. In other words, the benefit for one panel should not be significantly less than the benefit for another.

No.	Mission Name	Cost	TRL	HHS	LEB	SE	CVC	WSA	WR
1	CLARREO A	300	2013	0.018	0	0	0.144	0.078	0.028
2	CLARREO B	300	2013	0.018	0	0	0.144	0.078	0.028
3	DESDynI B	300	2011	0	0.070	0.169	0.072	0	0.016
4	GPSRO	230	2011	0.037	0.193	0.117	0.112	0	0.029
5	SMAP	450	2010	0.005	0.035	0.021	0.106	0	0.058
6	HYSPIRI A	350	2011	0	0	0.015	0	0	0.019
7	HYSPIRI B	350	2013	0	0	0.015	0	0	0.019
8	GEOCAPE B	400	2014	0.096	0.164	0	0.126	0.066	0.002
9	GEOCAPE C	400	2014	0.056	0.043	0	0	0.066	0
10	ACE B	300	2013	0.107	0.285	0.022	0.129	0.132	0.026
11	LIST A	600	2016	0.014	0	0.063	0.035	0	0.065
	Depreciation Rate			0.90	0.90	0.95	0.85	0.90	0.90

Table 3.5: The names, costs, TRL dates and benefits for each panel of the 11 possible early missions.

Since the benefits depreciate yearly, it does not make much sense to calculate the solutions for a time span of 40 year because the benefits will diminish after the first decade. Also, it is not realistic to arrange a mission 30 years ahead of time. Thus, I only model the problem for the first ten years, i.e., from 2011 to 2020. Moreover, from the observation of the original data table in [35], we can identify a subset of 11 missions that are most beneficial and are preferable to be launched in the first 10 years. The 10 missions are the ones shown in Table 3.5.

I again illustrate the formulation of the problem based on the four elementary parts of COPs: decision variables. a finite domain for each decision variable, constraints

and objectives. The decision variable x_i represents the launching year of mission i. For example, x_1 is the launching year of the mission "CLARREO A". Obviously, the domains for each x_i should be from 2011 to 2020. However, because of the TRL constraint, some options can be discarded at this stage, rather than leaving them in the constraints. For instance, the options for x_{11}, or mission "LIST A" can be reduced to 2016, 2017, 2018, 2019, 2020, instead of 2011..2020, because its TRL year is 2015. In sum, let TRL_i denote the TRL date of mission i, we have:

$$x_i = \{TRL_i \ .. \ 2020\}, \qquad i = 1, 2, \cdots, 11$$

The constraints are all related to the budget. Note that all the missions cost no more than the total budget for two years. Using this observation, we can cluster two years together when a mission costs more than 300 million dollars. For example, the constraints related to "SMAP" (x_5) are:

$$x_5 \neq x_i, \qquad i = 1, \cdots, 4, 6, \cdots, 11$$

$$x_5 \neq x_i + 1, \qquad i = 1, \cdots, 4, 6, \cdots, 11$$

The first constraint means that mission SMAP needs to use all the budget of the year it is launched, thus, no other mission should be scheduled to launch in the same year. The second constraint means that mission SMAP also takes the budget in the year before it is launched, thus, no mission should be scheduled in that year as well. This rule applies to missions HYSPIRI A, HYSPIRI B, GEOCAPE B, GEOCAPE C, and LIST A as well. An exception is that GPSRO and the HYSPIRI A can be launched in consecutive years, because the total cost of the the two missions is 580 million, which is less than the budget of two years. The exception is also valid for GPSRO and HYSPIRI B.

There are two objectives needed to be optimized: one is the total benefits, and the other is the fairness. When considering benefits, we can use the following formulation

to calculate the benefits obtained by launching satellite i:

$$B_m = \sum_{p=1}^{6} b_{mp} \cdot d_p^{(x_m - 2010)}, \qquad m = 1, 2, \cdots, 11$$

where b_{mp} represents the benefit of launching mission m for panel p, and d_P represents the benefit depreciation rate of panel p. The formulae sums up all the depreciated benefits in year x_m for each panel, which is the total benefit of launching mission m in year x_m. By summing up the benefits of all the missions, we can obtain the total benefits, i.e., the total benefit is:

$$B = \sum_{m=1}^{11} B_m$$

For the fairness objective, on the other hand, we can sum up the maximum gap of the benefits for each panel when a mission is launched. The unfairness of launching satellite m can be represented by the following formulae:

$$F_m = \max_{p_1, p_2 = 1 \cdots 6} \{b_{mp_1} \cdot d_{p_1}^{(x_m - 2010)} - b_{mp_2} \cdot d_{p_2}^{(x_m - 2010)}\}, \qquad m = 1, 2, \cdots, 11$$

where b_{mp_1} represents the benefit of launching mission m for panel p_1 and b_{mp_2} represents the benefit of launching mission m for panel p_2. The term $b_{mp_1} \cdot d_{p_1}^{(x_m - 2010)} - b_{mp_2} \cdot d_{p_2}^{(x_m - 2010)}$ represents the gap of benefits (after depreciation) between panels p_1 and p_2 if satellite m is launched in year x_m. By finding the maximum gap of all possible combination of p_1 and p_2, we can obtain the maximum gap of of benefits between any two panels, which is defined as the unfairness of launching satellite m. The total unfairness is the sum of the unfairness of all the missions, i.e., the total unfairness is:

$$F = \sum_{m=1}^{11} F_m$$

Having all the four elemental parts of an COP in hand, we can integrate them to form a decadal survey COP and solve it using the algorithms introduced in Chapter

2. The model described above looks simple because of the compactness of the mathematical symbols. In the real COP model, however, all the terms that are abbreviated by the dots need to be written down explicitly. For instance, to express $x_1 \neq x_2$, I need to write from ((NOT $x_1 = 2013$) OR (NOT $x_2 = 2013$)) to ((NOT $x_1 = 2020$) OR (NOT $x_2 = 2020$)), which costs eight lines of codes. The final model spans almost 8000 lines.

The results obtained by solving the decadal COP are shown in Figure 3-5. Seher's result[35] is also included as a comparison. The Pareto plot is shown in Figure 3-6. The X axis represents the degree of unfairness and the Y axis represents the total benefits. Points 1 to 7 correspond to Solutions 1 to 7, respectively. Point S corresponds to Seher's result (We call it Solution S). Since we maximize the total benefits and minimize the unfairness, the Pareto front is convex pointing to the upper-left corner.

	Seher's Result		Solution 1		Solution 2		Solution 3	
	Launch Date	Sequence	Launch Date	Sequence	Launch Date	Sequence	Launch Date	Sequence
Clarreo A	2015	4	2014	4	2015	4	2019	7
Clarreo B	2016	5	2015	5	2014	5	2020	8
DESDynI B	2013	2	2011	1	2011	1	2011	1
GPSRO	2011	1	2012	2	2012	2	2012	2
SMAP	2030	14	-	-	2019	8	2018	6
HYSPIRI A	2023	10	-	-	-	-	-	-
HYSPIRI B	2017	6	2016	6	2016	6	2016	5
GEOCAPE B	2020	8	2017	7	2018	7	2015	4
GEOCAPE C	2027	12	-	-	-	-	-	-
ACE B	2014	3	2013	3	2013	3	2013	3
LIST A	2019	7	2019	8	-	-	-	-
	Solution 4		Solution 5		Solution 6		Solution 7	
	Launch Date	Sequence	Launch Date	Sequence	Launch Date	Sequence	Launch Date	Sequence
Clarreo A	2020	7	2019	7	2017	5	2017	5
Clarreo B	2019	8	2020	8	2016	6	2016	6
DESDynI B	2011	1	-	-	-	-	-	-
GPSRO	2012	2	2011	1	2011	1	2012	2
SMAP	2018	6	2018	6	2019	7	2019	7
HYSPIRI A	-	-	2012	2	2012	2	2011	1
HYSPIRI B	2016	5	2016	5	2015	4	2015	4
GEOCAPE B	-	-	-	-	-	-	-	-
GEOCAPE C	2015	4	2015	4	2014	3	2014	3
ACE B	2013	3	2013	3	2020	8	2020	8
LIST A	-	-	-	-	-	-	-	-

Figure 3-5: Solutions of the decadal survey study. It includes Seher's solution and the solutions obtain by solving the multi-objective COP model introduced in this section.

From Figure 3-5, we can see that Solution 1 is the closest to Solution S. The launching years for each satellite are almost the same except the exchange of DESDynI B and GPSRO, and GEOCAPE B and LIST A. In Figure 3-6, Solution 1 corresponds

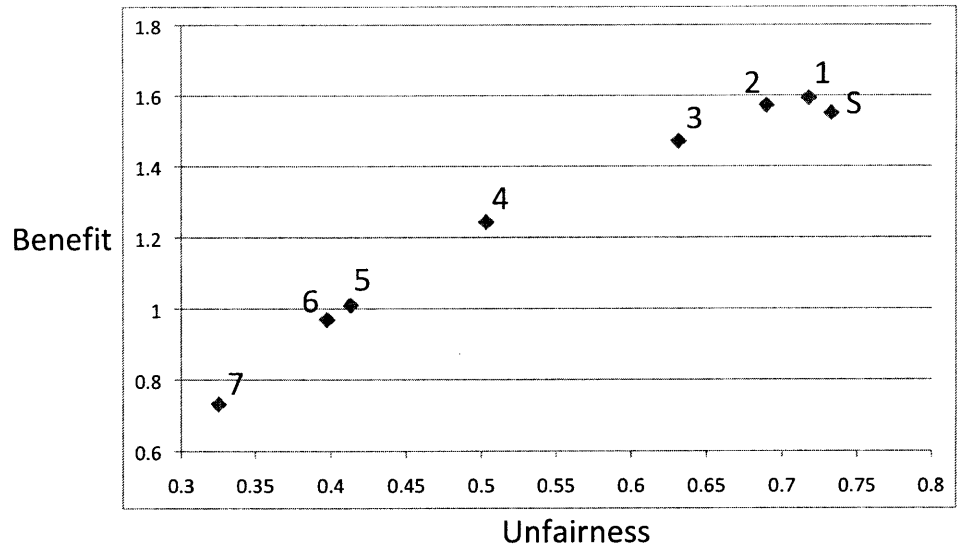


Figure 3-6: Plot of the results on the convex Pareto front. The Y axis represents the total benefits for all 6 panels by launching the satellites. The X axis represents the sum of the maximum gaps between panels by launching each satellite.

to Point 1, which has the best total benefits, but is also the most unfair one among all the solutions. It is also the closest to Point S, which represents Solution S. Point 7 represents the fairest mission sequence, but also has the lowest total benefit. Points 2 to 6 lie in the middle, representing the solutions that have some trade-off between the total benefit and the fairness concern. The Pareto plot shows that the points almost lie on a straight line, which means the gain on one objective offsets the loss on the other. Under such circumstance, points 2, 3 and 4 might be good alternatives, since they achieves a certain degree of fairness, while not losing too much in the total benefits.

Solution	1	2	3	4	5	6	7
Year	0.91	2.73	3.36	4.09	4.09	4.27	4.73
Sequence	0.36	1.91	2.27	2.82	3.00	3.18	3.18

Table 3.6: The difference between Seher's solution and Solutions 1 to 7. The value represents the average year and sequence gap of each mission between solutions.

Table 3.6 shows the average difference of launching year and launching sequence between Solutions 1 to 7 and Solution S. The average difference of launching year between Solution 1 and Solution S, for example, is calculated by averaging the absolute

value of the difference between the launching year of each mission in Solution 1 and Solution S. More precisely, assuming x_{sm} represents the launching year of mission m in Solution S, while x_{1m} represents the launching year of mission m in Solution 1, the average difference of launching year is calculated by the formulae:

$$D_{s1} = \frac{1}{11} \sum_{m=1}^{11} |x_{sm} - x_{1m}|$$

Correspondingly, we can calculate $D_{s2}, D_{s3}, \cdots, D_{s7}$. The average difference represents the difference between one solution to another to some extent. From Table 3.6, we can see that Solution 1 is the closest to Solution S. In average, each satellite is scheduled to be launched either about a year earlier or a year later in Solution 1 than planned in Solution S. The difference gradually increases from Solution 2 to Solution 7. Finally in Solution 7, each mission is launched nearly 5 years later or earlier than scheduled in Solution S. Considering the 10 years horizon, we can see that Solution 7 adopts a very different sequence from Solution S.

In this section, I introduce the background of earth science decadal survey study. The study involves a decision-making problem in which decision makers need to determine the launching sequence of a series of satellites that help scientists observe the earth and carry out experiments. I illustrate in detail how to model the problem with multi-objective COP, and explain the results obtained by solving the COP. I finally compare my solutions and Seher's result [35].

3.3 Summary

This chapter provides two case studies that use multi-objective COP to model practical problems. The first case study is the Apollo mission mode design. I adopt nine key decisions that have direct impact on the mission's success as the decision variables. I explain the meaning of the options for each decision variable and the constraints that limit the relationships of the variables. Moreover, I discuss the objectives that are needed to be optimized and show the final results, which coincide with Simmons'

results [36]. I also discuss the practical meaning of the points on the convex Pareto front obtained by solving the multi-objective COP. This case study shows that we can use multi-objective COP to model practical decision-making problems, and also shows the capability of the algorithms introduced in Chapter 2 to find the convex Pareto front as well as the solutions in an area that is close to the convex Pareto front.

The second case study is the earth science decadal survey study. The goal of the decadal survey is to decide an optimal sequence of launching a series of satellites that are used for the research of earth observation. I encode the problem by using a set of decision variables that denote the years when the satellites are launched. Based on this set of variables, I demonstrate how to express the budget constraint. I also explain how to express the two important objectives in multi-objective COP: the total benefits and the fairness concern. Finally, I show the results of the problem in addition to a plot of the convex Pareto front, and I compare the results to Seher's solution [35].

From the two case studies, we can see that the modeling skill and the algorithms introduced in Chapter 2 is applicable to the real-world decision-making problems. It supports the claim that many problems in the real world can be expressed as multi-objective COPs and then solved efficiently by the multi-objective COP solver using the algorithm developed in Chapter 2.

Chapter 4

Optimization Methods for System Architecture

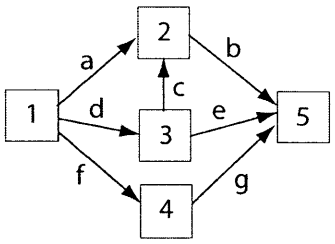
This chapter introduces and develops methods for solving two main problems. The first is how the DSM and OPD are connected and how to translate between the two system representation methods. The second is how to cluster a single DSM, and multiple DSMs as well. Section 4.1 introduces the characteristics of DSM and OPD, as well as the complementary of their characteristics. In Section 4.2, I reveal the Markov property of the DSM description of OPD, and demonstrate the projection relation between them. In Section 4.3, I introduce the DSM clustering problem and describe the specific goals mathematically. Then I provide an optimization model which encodes the problem and show the clustering results of an pedagogical example and a practical problem. I also develop an algorithm to solve multiple DSM clustering in this section. Finally, I summarize the whole section in 4.4.

4.1 The Complementary Characteristics of DSM and OPD

4.1.1 Characteristics of DSM

As mentioned in Section 1.1.3, Design Structure Matrix (DSM) provides an effective approach for designing and managing complex engineering projects and systems. Instead of graphical representation used in traditional system modeling tools such as PERT and CPM, it describes the dependencies and feed-backs between different parts in systems with a simple and compact matrix. It is also efficient for system decomposition and integration, as discussed in Section 1.1.3. Because of the advantages of DSM, it is now becoming more and more popular for various purposes in industrial design and management such as product development, project planning, project management, system engineering, organization design, etc[2]. Eppinger provides several typical applications of DSM in project management[19].

Classical DSMs map directed system graphs to matrices in which the entries (i, j) are marked by a check sign if there exist edges pointing from node j to node i. Diagonal entries are either blacked out or used to capture special information of the corresponding part in the system[44]. Figure 4-1 shows a simple illustration of the relationship between system graph representation and DSM.



	1	2	3	4	5
1					
2	✓		✓		
3	✓				
4	✓				
5		✓	✓	✓	

Figure 4-1: An example of the classical Design Structure Matrix (DSM) representation of a system.

Simple as DSM is, it only represents a restricted class of system relationship. It is unable to capture miscellaneous relationship between nodes by merely using a check sign. Information of causal relationship is usually classified into several categories.

However, a specific DSM is usually created to represent a single type of causal relationship, such as the relationship of processes with other processes. Objects, both instruments and operands, are ignored or implicit. Moreover, relationships between nodes in a system are not necessarily causal. For example, transmission itself does not change speed, while it serves as an instrument of changing speed smoothly. A more fundamental limitation of the DSM is its difficulty in representing relationships that are not binary – not just connecting two elements at a time, but linking three or more elements. These challenges of DSMs are easily solved by a bi-partite graph representation of Object-Process Diagram (OPD).

4.1.2 Characteristics of OPD

Object-Process Diagram (OPD) provide a bi-partite graphical tool to represent a very general complex system of operand objects, processes and instrument objects[15]. The OPD representation can be used to explicitly represent all causal or non-causal relationships within a system. OPDs are built up of objects and processes. Objects are things that exist or have the potential for existence, and have states. Processes are transformations that can change the states of objects. Objects are further divided into instrument objects and operands. The distinction is that the instrument is the agent of the process, while the operand is the object whose states are affected by the process.

A simple and generic OPD is shown in Figure 4-2, where each rectangle denotes an object and each oval denotes a process. In this OPD, the bottom instrument is an agent of the process (round headed arrow), the left operand creates the process (single headed arrow leading to the process), and the right operand is created by the process (single headed arrow leading away from the process). One operand, one process, and one instrument object is the canonical structure of a system (as it is a sentence in natural human language), and all complete descriptions of systems must have these elements and their interrelationships.

Although OPD is a powerful tool to model complex systems, to date, it does not have a complementary matrix representation that could facilitate the computation.

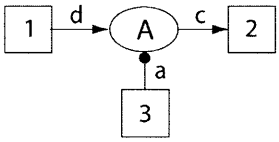


Figure 4-2: A generic Object-Process Diagram (OPD)

In the case of modeling large-scale system, OPD will be difficult to be fully comprehended since it may be as lengthy as hundreds of pages, especially when it involves large amount of information exchange across the whole system. In such cases, DSM would be very helpful to decompose the system and clarify the relationships between subsystems.

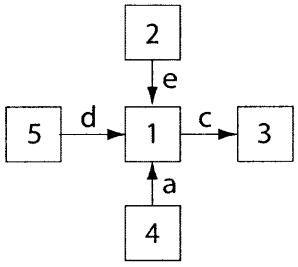
Since the advantages and disadvantages of the two system representations are complementary, we naturally seek to link the DSM and the OPD together. With the DSM representation, we can translate OPDs into DSMs and apply computation tools on them. The DSM representation of OPD also enables us to condense the OPDs to simplified structures, which is explained in Section 4.2.3.

4.2 Relation between Graph Representation and Matrix Representation of Systems

In this section, I first illustrate an extension of the classical DSM and show the correspondence between the general graphical system representation and the DSM. It is followed with a demonstration of the Markov property of the DSM. Secondly, I explain the concept of "projection" from the general graphical system representation to the DSM with several examples. Finally, I derive the DSM representation of OPD and reveal the special structure of this class of DSM, of which we can take advantage to promote the computational efficiency. In the summary subsection, the limitations and possible extensions of the DSM representation are discussed.

4.2.1 The Extension of DSM and its Markov Property

In the classical DSM, people simply put a check mark in the entries of the matrix. However, in most of the complex systems, the relationships between different parts vary from one to another. System architects need to preserve and analyze such information. Here, I extend the DSM to preserve more information regarding to the relationship between a pair of nodes. Figure 4-3 defines a new rule of translating from a system to an extended DSM.



Classical DSM:

	1	2	3	4	5
1		✓		✓	✓

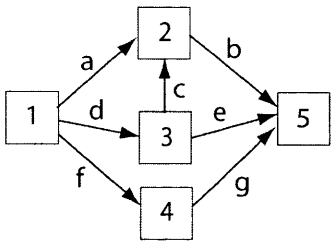
Extended DSM:

	1	2	3	4	5
1		e'	c	a'	d'

Figure 4-3: Comparison of the definitions of the extended DSM and the classical DSM.

As presented in Figure 4-3, the extended DSM represents the relation between node 1 and its neighbor nodes. In the traditional DSM convention, the entry (i, j) represents the relation from node j to i. However, here I define the entry (i, j) to be the relation from node i to node j. We will see the reason later in this section. If the relation from i to j is along the direction of the arrow, we put a lowercase letter in the entry (i, j). If it is in the reverse direction, we add a prime to the letter. Figure 4-4 shows an complete extended DSM for the system in Figure 4-1. Especially, we assign 0 to the diagonal entries meaning that a node does not have self-loop edges.

Notice that the matrix is analogous to the one-step Markov chain (refer to [22] for the introduction to Markov chain if not familiar with it), except it is in the symbol form and the row sum is not necessarily 1. Despite of such a difference, we can still apply the multi-step transition property of Markov chain. Analogously, if D is a DSM matrix, D^n represents an n-step Markov transition relationship. I want to emphasize again that the diagonal entries of the DSMs are assumed to be 0 for now. I will



	1	2	3	4	5
1		a	d	f	
2	a'		c'		b
3	d'	c			e
4	f'				g
5		b'	e'	g'	

Figure 4-4: A complete extended DSM representation of a system.

discuss how to deal with the diagonal entries later. The 2-step transition matrix D^2 corresponding to Figure 4-4 is shown in Table 4.1.

	1	2	3	4	5
1		dc	ac'		$ab+de+fg$
2	$c'd'$		$a'd+be'$	$a'f+bg'$	$c'e$
3	ca'	$d'a+eb'$		$d'f+eg'$	cb
4		$f'a+gb'$	$f'd+ge'$		
5	$b'a'+e'd'+g'f'$	$e'c$	$b'c'$		

Table 4.1: A two-step transition DSM corresponding to Figure 4-4. It shows all the possible two-step paths between each pair of nodes.

This matrix contains all possible 2-step paths between any pair of nodes. For instance, the entry $(1,5)$ means there are three paths from node 1 to 5 in 2 steps: ab, de and hg. From node 3 to 4, there are 2 paths, $d'h$ and eg', respectively. Notice that the paths include backward flows. The 3-step transition DSM is shown in Table 4.2.

Before I proceed, I would like to explain why I do not adopt the DSM convention which reads the column first then the row. In the matrix shown in Table 4.1, when we want to check the path from node 1 to 2, it is $1dc2$, clearly indicating the path 1-d-3-c-2. If we used the DSM convention, the entry $(1,2)$ would be $d'c'$. Naturally, people will understand it as the case that there is a path $2d'c'1$. However, there is no such a path existing in the graph. Actually, we have to read the content in the entry reversely, that is, $c'd'$ instead, and a path $2c'd'1$ exists in the graph (2-c'-3-d'-1). Due to this awkwardness, I use the first-row-then-column convention instead throughout this whole chapter.

One might have noticed that although the matrix D^n represents the n-step transition relationship between each pair of nodes in the system, it is problematic when the

	1	2	3
1		$ac'c + (ab + de + fg)b'$	$dcc' + (ab + de + fg)e'$
2	$(a'd + be')d' + (a'f + bg')f'$		$c'd'd + c'ee'$
3	$(d'a + eb')a' + (d'f + eg')f'$	$ca'a + cbb'$	
4	$(f'a + gb')a' + (f'd + ge')d'$	$(f'd + ge')c$	$(f'a + gb')c'$
5	$e'ca' + b'c'd'$	$(b'a' + e'd' + g'f')a + b'c'c$	$(b'a' + e'd' + g'f')d + e'cc'$

	4	5
1	$(ab + de + fg)g'$	$dcb + ac'e$
2	$c'd'f + c'eg'$	$(a'd + be')e + (a'f + bg')g$
3	$ca'f + cbg'$	$(d'a + eb')b + (d'f + eg')g$
4		$(f'a + gb')b + (f'd + ge')e$
5	$(b'a' + e'd' + g'f')f$	

Table 4.2: A three-step transition DSM corresponding to Figure 4-4. It shows all the possible three-step paths between each pair of nodes.

path includes immediate backward flow. For example, the 3-step transition relationship from node 1 and node 2 includes the path $ac'c$, while obviously, the consecutive sub-path $c'c$ is redundant. Such redundancy will inflate as the order of D^n increases. Naturally, we want to discard the $c'c$ sub-path; however, we will end up with a 1-step path a, which makes D^3 a pseudo 3-step transition DSM.

To avoid such ambiguity, we need to revise the original definition of extended DSM and the concept of n-step transition matrix slightly. We introduce a new concept – Modified N-Step DSM $D^{(n)}_{mod}$, as a representation of all possible paths between each pair of nodes that linking each other WITHIN n steps. Before discussing the modified n-step DSM, I need some extra algebraic rules for symbols. In the following rules, I is either 1 or an identity matrix, depending on the type of the symbols. It potentially means the node is reached with 0 step.

Rule 1 $\forall x,\ x'x = xx' = I.$ (e.g. $c'c = I$)

Rule 2 $nI = I.$ (e.g. $c'c + aa' = I + I = 2I = I$)

Rule 3 Terms in which a symbol appears twice are eliminated.

Rule 4 Force all diagonal entries equal to I.

The key intention of rules 1 and 2 is to eliminate the duplications in the n-step DSM. Rule 3 eliminates potential cycles. For example, assume the (i, j) entry of the

DSM is a_{ij}. When the order of D is high enough, one possible term of the $(1,2)$ entry in the DSM is $a_{12} = a_{15} \cdot a_{52} = ab \cdot g'f'a = abg'f'a$, in which $abg'f'$ is a cycle that is redundant. Without rule 3, it will stay in higher order results. But according to rule 3, since the symbol a appears twice in this term, it will be eliminated, and thus the cycle will not appear in the modified n-step DSM. Rule 4, which is different from our original definition of extended DSM, adds a self loop to each node. The modified 1-step to 3-step DSMs are shown in Tables 4.3, 4.4, and 4.5, respectively.

	1	2	3	4	5
1	I	a	d	f	
2	a'	I	c'		b
3	d'	c	I		e
4	f'			I	g
5		b'	e'	g'	I

Table 4.3: An example of Modified 1-Step DSM. It is the same as the original 1-Step DSM, except having I in each diagonal entry, representing a self-loop.

	1	2	3	4	5
1	I	$a + dc$	$d + ac'$	f	$ab + de + fg$
2	$a' + c'd'$	I	$c' + a'd + be'$	$a'f + bg'$	$b + c'e$
3	$d' + ca'$	$c + d'a + eb'$	I	$d'f + eg'$	$e + cb$
4	f'	$f'a + gb'$	$f'd + ge'$	I	g
5	$b'a' + e'd' + g'f'$	$b' + e'c$	$e' + b'c'$	g'	I

Table 4.4: An example of Modified 2-Step DSM. It shows all possible paths from one node to another within 2 steps.

There is a close relationship between the extended DSM and the modified n-step DSM. Intuitively, if node A can reach node B within n steps, the corresponding path will show up in the n-step transition DSM. That is, the path with less than n step will be present in D^n_{mod}. Formally, regarding to the modified n-step DSM, we have the following theorem:

Theorem 1 A modified n-step DSM equals the sum of all the extended k-step DSMs, in which $0 \leq k \leq n$.

Proof: *Since the diagonal entries of the extended n-step DSM are all 0, we can write*

	1	2	3
1	I	$a + dc + deb' + fgb'$	$d + ac' + abe' + fge'$
2	$a' + c'd' + be'd' + bg'f'$	I	$c' + a'd' + be'$
3	$d' + ca' + eb'a' + eg'f'$	$c + d'a + eb'$	I
4	$f' + gb'a' + ge'd'$	$f'a + gb' + f'dc + ge'c$	$f'd + ge' + f'ac' + gb'c'$
5	$b'a' + e'd' + g'f' + e'ca' + b'c'd'$	$b' + e'c + e'd'a + g'f'a$	$e' + b'c' + b'a'd + g'f'd$
	4	5	
1	$f + abg' + deg'$	$ab + de + fg + dcb + ac'e$	
2	$c'd'f + c'eg'$	$b + c'e + a'de + a'fg$	
3	$d'f + eg' + ca'f + cbg'$	$e + cb + d'ab + d'fg$	
4	I	$g + f'ab + f'de$	
5	$g' + b'a'f + e'd'f$	I	

Table 4.5: An example of Modified 3-Step DSM. It shows all possible paths from one node to another within 3 steps.

the modified 1-step DSM in the form $D_{mod} = (D + I)$.

$$\begin{aligned} D_{mod}^{(n)} &= D_{mod}^{n} = (D + I)^n \\ &= \sum_{k=0}^{n} \binom{n}{k} \cdot D^k \\ &= \sum_{k=0}^{n} D^k \end{aligned}$$

The equality in the third line holds because $\forall t \in \mathbb{Z}$,

$$t \cdot D^k = tI \cdot D^k = I \cdot D^k = D^k$$

The second equity holds because of rule 2.

Further, since no symbol can appear twice in a single term, and the number of symbols in a DSM is finite, we can expect that the when n is large enough, the modified n-step DSM would reach a fixed point, that is, $D_{mod}^{(n+1)} = D_{mod}^{(n)}$. In this case, we have enumerated all possible paths between any two nodes in the system. In the example shown in figure 4-4, the fix point is reached in 4 steps. The modified 4-step DSM is too lengthy to be shown here, but one can verify that $D_{mod}^{5} = D_{mod}^{4}$, which

means within 4 steps, all the nodes can reach every other nodes.

In sum, from the discussions above, we the following conclusion: Under rules 1-4, the modified n-step DSM, $D_{mod}^{(n)} = D_{mod}^{n}$ represents all possible paths linking each pair of nodes within n steps. When n is large enough, DSM will reach a fixed point, i.e., $\exists\, n \geq 1,\ s.t.\ D_{mod}^{(n+1)} = D_{mod}^{(n)}$.

4.2.2 Projection from the Graphical Representation of Systems to DSM

In this part, I will explain what is a "projection" from a graphical representation to a DSM and how to project to DSM. In the previous example shown in Figure 4-4, if we view the node 1 as an input, node 5 as an output, nodes 2, 3 and 4 as the inner components of a system, sometimes engineers are only interested in the interaction between the input and output, that is, the interaction between nodes 1 and 5. In other words, only the small matrix

$$\begin{pmatrix} I & ab + de + hg + dcb + ac'e \\ b'a' + e'd' + g'h' + b'c'd' + e'ca' & I \end{pmatrix}$$

which includes all the possible paths from the input to the output, is needed. The interaction between the other parts are condensed out. The operation that maps a complete DSM to a sub-DSM is defined as a "Projection", which is derived from the concept of projection in linear algebra. If the projection maps the original DSM to a sub-DSM which contains all the paths between several nodes within n steps, we call the projection an n-step projection.

To implement the n-step projection, I introduce the projector π_p (m denotes the dimension of the matrix):

$$\pi_p = diag(a_1,\, a_2,\, \cdots,\, a_m), \quad a_i = \begin{cases} 1 & \text{node } i \text{ should be preserved} \\ 0 & \text{node } i \text{ should be condensed out} \end{cases}$$

Then the projection formula can be written as:

$$D_{\pi}^{(n)} = \pi_p \times D_{mod}^{n} \times \pi_p$$

where $D_{\pi}^{(n)}$ denotes the n-step sub-DSM after the projection, and D_{mod} denotes the modified 1-step DSM. Assume the number of nodes that we are interested in is k, and the number of all nodes is m. Instead of getting an $k \times k$ matrix, we will still obtain an $m \times m$ matrix, but with only $k \times k$ non-zero entries. For instance, the DSM shown in Figure 4-4 will end up being the matrix shown in Table 4.6 after the 3-step projection to nodes 1 and 5. In the same way, we can also calculate the 2-step and 4-step projections. One can verify that at step 4, D_{π} reaches a fixed point, or equivalently $D_{\pi}^{(4)} = D_{\pi}^{(3)}$, which means none of the paths from node 1 to node 5 includes more than 3 steps, and we have obtained all the possible paths from node 1 to node 5. Table 4.7 shows another instance where we apply a 2-step projection to the DSM shown in Figure 4-2 to nodes 1, 3, and 5.

$D_{\pi_{15}}^{(2)} =$

	1	$2 \cdots 4$	5
1	I		$ab + de + hg + dcb + ac'e$
2			
$\vdots$		**0**	
4			
5	$b'a' + e'd' + g'h' + b'c'd' + e'ca'$		I

Table 4.6: An example of a 3-step projection from the DSM shown in Figure 4-2 to nodes 1 and 5. It contains the paths linking nodes 1 and 5 within 3 steps.

$D_{\pi_{135}}^{(2)} =$

	1	2	3	4	5
1	I		$d + ac'$		$ab + de + hg$
2					
3	$d' + ca'$		I		$e + cb$
4					
5	$b'a' + e'd' + g'h'$		$e' + b'c'$		I

Table 4.7: An example of a 2-step projection from the DSM shown in Figure 4-2 to nodes 1, 3 and 5. It contains the paths linking nodes 1, 3 and 5 within 3 steps.

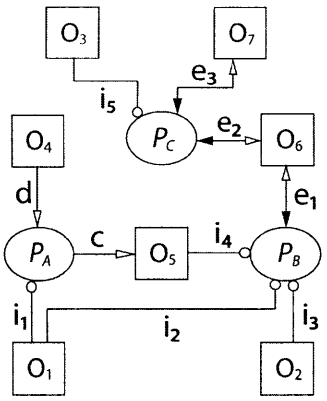
If we calculate the n-step transition DSM following the formula strictly, the algorithm will take $c \cdot n \cdot [m^3 + o(m^2)]$ computational time, in which c is a constant and

$\lim_{m\to\infty} \frac{o(m^2)}{m^3} = 0$. When implementing the projection, however, we do not need to go through all the matrix multiplications. First, if we view x' as x^T, which implies $y'x' = y^T x^T = (xy)^T$, then the DSM is symmetric, which means we only need to calculate the upper triangular matrix. This simplification will reduce the constant c to $c/2$. Moreover, starting with the left projection, we shall notice that the rows corresponding to the nodes that are condensed out keep being 0 through the computational process. Making use of this feature, we can avoid the computation of these rows in the matrix multiplication. In addition, at the last step, we do not have to multiply π_p, but just discard the unwanted columns and rows, and extract the final result of the projection. Using the time-saving tricks provided above, we can reduce the total computational time to $\frac{c}{2} \cdot n \cdot [km^2 + o(m^2)]$, where k is the number of nodes that we need in the DSM after the projection.

4.2.3 Projection from OPD to DSM

As mentioned in Section 4.1, OPD is one of the important graphical representation tools. It has a characteristic that does not exist in generic system graphs: Since one can only link objects to processes in OPDs (no object to object or process to process arc), OPD is a bi-partite graph [15]. (A graph is bi-partite if and only if the nodes in the graph can be divided into two groups, and the arcs in the graph only exist between the two graphs, but not within the groups). Obviously, OPD can be divided into two groups - objects and processes, and the nodes only connect to the nodes in the other group. The bi-partite property of OPD is very special, because the DSM corresponding to an OPD has only non-zero entries in entries linking an Object and a Process. I am therefore able to make use of such property to further simplify and accelerate the computation. An OPD and its corresponding DSM are shown in Figure 4-5.

As we can see in Figure 4-5, because of the bi-partite property of the OPD, the corresponding DSM only has non-zero entries on the bottom-left corner and top-right corner, which are the parts representing the relationship between objects and



	A	B	C	1	2	3	4	5	6	7
A				i'_1			d'	c		
B				i'_2	i'_3			i'_4	e'_1	
C						i'_5			e'_2	e'_3
1	i_1	i_2								
2		i_3								
3			i_5							
4	d									
5	c'	i_4								
6		e_1	e_2							
7			e_3							

Figure 4-5: An example of the DSM representation of an OPD. The DSM has a special structure that only the parts that represents the relation between objects and processes are not empty.

processes. Generally, the DSMs corresponding to OPDs have the following structure:

$$\begin{bmatrix} \mathbf{0} & Q^T \\ Q & \mathbf{0} \end{bmatrix}$$

where Q is the matrix representing the relationships between objects and processes. The matrix Q, which is a part of the complete DSM of the objects and processes, is also called Multiple Domain Matrix (MDM)[26]. If we assume the numbers of processes and objects in the OPD are p and q respectively, the dimension of Q is $q \times p$.

Now, I can show that not only the structure of the DSMs representing OPDs is special, but also the structures of exponents of D:

$$D^{2n} = \begin{array}{|c|c|} \hline (Q^TQ)^n & \mathbf{0} \\ \hline \mathbf{0} & (QQ^T)^n \\ \hline \end{array} \qquad D^{2n+1} = \begin{array}{|c|c|} \hline \mathbf{0} & (Q^TQ)^nQ^T \\ \hline (QQ^T)^nQ & \mathbf{0} \\ \hline \end{array}$$

Such a result is intuitive: in even steps, only pairs of objects or pairs of processes can be connected. On the other hand, in odd steps, only pairs of nodes with exactly one object and one process could be connected.

In both cases, the main region that we need to compute is the upper triangular part of $(Q^TQ)^n$ or $(QQ^T)^n$, since the DSM is symmetric. Which one of the two forms requires less computational time? It depends on their dimensions. The dimension of $(Q^TQ)^n$ is $p \times p$, while the dimension of $(QQ^T)^n$ is $q \times q$. Hence, if $p \leq q$, we choose the former one, otherwise the latter one. Then the computation time for n-step transition DSM is

$$\frac{c}{2} \cdot \left\lfloor \frac{n}{2} \right\rfloor \cdot \min(p, q)^3$$

The n-step transition DSM calculated in the form of Q is still very complex because of much redundancy of immediate backward flow. Recall the rules 1-4 and the concept of modified n-step DSM in the last section. We have the following corollary under rules 1-4:

Corollary 1 The modified $2n$-step DSM of OPD $D_{mod}^{(2n)}$ has the structure $\begin{bmatrix} D_1 & D_2 \\ D_3 & D_4 \end{bmatrix}$,

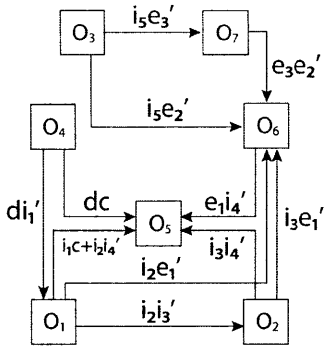
where $D_1 = \sum_{k=1}^{n} (Q^TQ)^k$, $D_2 = \sum_{k=1}^{n} (Q^TQ)^k Q^T$,

$D_3 = \sum_{k=1}^{n} (QQ^T)^k Q$, $D_4 = \sum_{k=1}^{n} (QQ^T)^k$

The corollary brings down the computational time of the modified n-step DSM to the level of the n-step transition DSM, i.e., $O\left(\left\lfloor \frac{n}{2} \right\rfloor \cdot \min(p, q)^3\right)$. Instead of computing the complete D_{mod}^{2n}, we can first compute the sum of $(Q^TQ)^k$ or $(QQ^T)^k$ from 1 to $2n$, then adjust the result according to the formulas D_1 to D_4 above, and finally put all the four parts together.

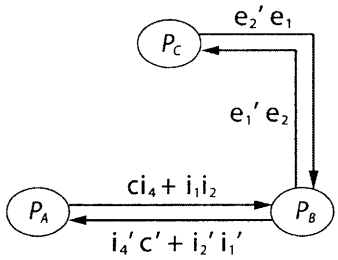
I shall close this section by showing two projection examples from the OPD in Figure 4-5 to DSMs. Assume one only care about the 2-step transition relations between all the objects, which means we shall condense out all the processes. The approach of solving this problem is straightforward: we only need to extract the D_4 part from the modified 2-step DSM, i.e., $D_4 = \sum_{k=1}^{n} (QQ^T)^k$, since it represents the relation between objects. The result is shown in Figure 4-6. One can find all the 2-step paths from one object to another in the table. Similarly, we can also condense out all the objects and get the 2-step relations between processes as well. The result

is shown in Figure 4-7.



	1	2	3	4	5	6	7
1	I	i_2i_3'		i_1d'	$i_1c+i_2i_4'$	i_2e_1'	
2	i_3i_2'	I			i_3i_4'	i_3e_1'	
3			I			i_5e_2'	i_5e_3'
4	di_1'			I	dc		
5	$c'i_1'+i_4i_2'$	i_4i_3'		$c'd'$	I	i_4e_1'	
6	e_1i_2'	e_1i_3'	e_2i_5'		e_1i_4'	I	e_2e_3'
7			e_3i_5'			e_3e_2'	I

Figure 4-6: A 2-step projection to objects. The tables only contains the paths connecting one object to another. No information of processes is maintained.



	A	B	C
A	I	$ci_4+i_1i_2$	
B	$i_4'c'$	I	$e_1'e_2$
C		$e_2'e_1$	I

Figure 4-7: A 2-step projection to processes. The tables only contains the paths connecting one process to another. No information of objects is maintained.

4.2.4 Summary

In this section, I first extend the classical DSM to enable it to store different relations between nodes. Based on the extension, I reveal the Markov property of the DSM and derive the formulation of n-step transition DSM. However, the n-step transition DSM is not decently defined if the immediate back-flow is eliminated. To avoid such confusion, I introduce the modified n-step DSM along with four algebraic rules. The modified n-step DSM represents all possible paths between the nodes within n steps.

Further, to provide a tool for information condensation, I introduce the concept of projection. The projection from the graphical representation of systems to the DSM is a useful way to compress both the graph and the matrix and let system

architects concentrate on a part of the matrix that they are interested in. I also show the evidence that we can accelerate the computation when projection is applied.

After the discussion of the general projection from graphical representation of systems to DSM, I continue with the discussion of the bi-partite property of OPD, and illustrate the special structure of the DSM corresponding to the OPD. I also introduce how to further simplify and accelerate the computation utilizing such a special structure. I finally demonstrate the projection on OPD with two examples in which we are only interested in objects or processes.

Since the extended DSM has some good linear-algebraic properties, such as symmetry and $xx' = x'x = I$, we can further develop algorithms to make full use of such properties. For instance, if we are able to develop a computable eigenvalue decomposition on DSM $D = Q^*\Lambda Q$, where Q is unitary matrix and Λ is diagonal matrix, we will be able to compute $D^n = Q^*\Lambda^n Q$, which will be a significant promotion on the overall computational efficiency.

4.3 Optimization Methods for DSM Clustering

As mentioned in Sections 1.2.3 and 4.1, the DSM is very useful in complex system design and analysis. DSM allows engineers to decompose complex systems into manageable clusters, and integrate the clusters after they are analyzed or designed separately.

The standard process of using DSM to analyze complex systems is as follows. First, decompose a complex system into indivisible components. These components correspond to the rows and columns of the DSM. For example, in Figure 4-8, pedals, chain, gear shift, gears, etc. are the components. Secondly, combine the components into several clusters, and then design the system or divide the project team according to the clustering. For example, in Figure 4-8, the thick line divides the components into four clusters, that is, pedals and chain are in one cluster; gear shift, gears and wheels are in one cluster; brake and brake string are in one cluster; and handlebars and odometer are in one cluster. Note that there are still connections outside of

the cluster: that would represent the interface between clusters. For example, chain and gears are correlated, but they are not in the same cluster, and the connection between gears and chain is the interface between the padel-chain cluster and the gear shift-gears-wheels cluster. The final step is to integrate the whole system and analyze the overall properties.

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

Figure 4-8: A DSM describing the relations of bicycle components [31]. It also shows a good clustering result with the thick lines.

Most of the time, the first step is relatively easy, since in many systems, the basic components are already identified. For example, when designing a bicycle, the components like pedals, chain, brake, wheels, etc. are either standard components that could be fetched from other manufacturers directly, or are so small that no further decomposition is beneficial or necessary. There are, of course, some systems whose components are still unclear, but I do not intend to discuss such scenarios, since they vary in different systems. In the following process, I will assume the components are readily prepared.

The last step is to integrate the clusters together. This analysis is usually system dependent. Analyzing a rocket system is so different from analyzing an aeroplane system. Generally, there is no universal approach to analyze all complex systems.

The remaining step is the clustering of components. Fortunately, the clustering could be solved by universal approaches, regardless of the specific features of different systems, since it is essentially an optimization analysis on a matrix.

In this section, I will first illustrate the definition and goal of the DSM clustering problem. Then I will establish an integer programming model for the problem. Finally, I will demonstrate an approach to cluster multiple DSMs with same components

but different relations between components.

4.3.1 Optimal DSM Clustering

The goal of DSM clustering is to try to decompose the components into several clusters which have no or little connections with each other. Specifically, take a DSM with binary entry values as an example, there are four major objectives: (we say that components i and j are correlated if the (i, j) entry of the DSM is 1).

1. Maximize the number of 1's within the clusters. (Or equivalently, minimize the 1's out of the clusters.)

2. Minimize the number of 0's within the clusters. (Or equivalently, maximize the 0's out of the clusters.)

3. Minimize the number of correlated cluster pairs.*

4. Maximize the number of clusters.

* If a component in the i'th cluster is correlated with a component in the j'th cluster, we call clusters i and j correlated, and clusters i and j form a correlated cluster pair.

In the DSM clustering in Figure 4-8, there are nineteen 1's in the clusters, two 0's in the clusters. Cluster 1 (Pedals, Chain) and cluster 2 (Gear Shift, Gears, Wheels) form a correlated pair because component "Chain" in the first cluster and "Gears" in the second cluster are correlated. Also, the second and third clusters, second and fourth clusters are another two correlated pairs. Thus, there are three correlated cluster pairs in total.

Generally, objectives 1 and 3 are positively related, and objectives 2 and 4 are also positively related, while the objectives 1,3 and objectives 2,4 are negatively related. To see such a relation, let's consider two extreme cases as shown in Figure 4-9. Assume there are n components in total. When all the n components are in one cluster, as shown in the first DSM in Figure 4-9, the first and third objectives are optimized, since all the 1's are in the clusters and the number of correlated clusters is 0. But

objectives 2 and 4 are not optimized at all: all 0's are in the cluster, and the number of clusters is only 1. On the other hand, if every single component forms a cluster, as shown in the second DSM in Figure 4-9, there are n clusters in total, and objective 2 is optimized since no 0's are in the clusters. However, objectives 1 and 3 have the worst values.

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

Figure 4-9: Two extreme cases of DSM clustering. The first DSM clustering puts all components in one cluster. It maximizes the number of 1's in the cluster, and minimizes the number of correlated cluster pairs, but also maximized the number of 0's in the cluster. The second DSM clustering puts each component in one cluster. It minimizes the number of 1's in the clusters, and maximizes the number of correlated cluster pairs, but also minimizes the number of 0's in the clusters.

There is no conclusive assertion in the literature about how much attention should be paid to each of the objectives. In my model, I will use the following scheme: I will first fixed the number of clusters, and allow users to put different weights on each of the objectives 1, 2 and 3. The reason for fixing the number of clusters is that one can always start with a reasonable number of clusters, and then further decompose each of the clusters into several subgroups if the result is not satisfactory. The experimental results at the end of Section 4.3.2 shows that such a strategy is good enough to provide an answer that is not far from optimum.

Before I explain the DSM clustering method, I would like to mention that there may exist "bus" components in the DSM. The "bus" components are the ones that have connections to almost all of the other components. For example, when computers are assembled, the motherboard has interfaces with CPU, memories, sound cards, etc. Since the bus components are interfaces, we would like to identify and separate them before clustering the other components. Chen, L. and Li, S. [5] provide a way of distinguishing the bus components. In the following parts, I will assume that the bus components has been identified and excluded.

4.3.2 An Integer Programming Model for DSM Clustering

In this subsection, I will demonstrate the integer programming model for DSM clustering problems. The notation for the optimization model are shown in Table 4.8. The subscript i, j represents the components and k, l represents the clusters.

x_{ik}	-	A binary decision variable indicating whether to assign component i to cluster k.
c_{kl}	-	A binary variable indicating whether clusters k and l are correlated.
n_{ij}	-	A binary variable indicating whether components i and j are in different clusters.
z_{ij}	-	A binary variable indicating whether components i and j are in the same cluster and the entry (i, j) in the DSM is 0.
d_{ij}	-	The value of the entry (i, j) in the DSM.
w_1	-	The weight of the number of correlated cluster pairs (one of the three main goals).
w_2	-	The weight of the number of 1's within clusters (one of the three main goals).
w_3	-	The weight of the number of 0's within clusters (one of the three main goals).
N	-	The number of components.
G	-	The number of clusters.

Table 4.8: The notations used in the integer programming model for DSM clustering.

To further clarify the notations, let us take the clustering of the bike assembly DSM in Figure 4-8 as an example. Assume "Chain" is component 2, "Gear Shift" is component 3 and "Wheels" is component 5. Assume "Pedals, Chain" are in cluster 1, and "Gear Shift, Gears, Wheels" are in cluster 2. Then we have: $x_{21} = 1, x_{52} = 1$, since "Chain" is assigned to cluster 1, and "Wheels" is assigned to cluster 2. $c_{12} = 1$ since clusters 1 and 2 are correlated because of the correlation between components "Chain" and "Wheel". $n_{25} = 1$ since component "Chain" and "Wheels" are in

different clusters. $z_{35} = 1$ since components "Gear Shift" and "Wheels" are in the same cluster, but the DSM value $d_{35} = 0$.

Now we can illustrate the optimization model for the clustering problem. It is shown in Table 4.9. I will first explain each of the constraints and then go back to the objective function.

$$\min \quad w_1 \cdot \sum_{k=1}^{G}\sum_{l=1}^{G} c_{kl} + w_2 \cdot \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ij}n_{ij} + w_3 \cdot \sum_{i=1}^{N}\sum_{j=1}^{N} z_{ij}$$

$$\text{s.t.} \quad \sum_{k=1}^{G} x_{ik} = 1, \qquad \forall\, i = 1 \cdots N$$

$$\sum_{i=1}^{N} x_{ik} \geq 1, \qquad \forall\, k = 1 \cdots G$$

$$x_{ik} + x_{jl} + d_{ij} - 2 \leq c_{kl}, \qquad \forall\, i,j = 1 \cdots N, \quad \forall\, k,l = 1 \cdots G, k \neq l$$

$$x_{ik} + x_{jl} - 1 \leq n_{ij}, \qquad \forall\, i,j = 1 \cdots N, \quad \forall\, k,l = 1 \cdots G, k \neq l$$

$$x_{ik} + x_{jk} + (1 - d_{ij}) - 2 \leq z_{ij}, \qquad \forall\, i,j = 1 \cdots N, \quad \forall\, k = 1 \cdots G$$

Table 4.9: The integer programming model for DSM clustering.

The first constraint simply means that each component i should be and can only be assigned to one cluster.

The second constraint means that each cluster k should contain at least one component. This inequality eliminates empty clusters. In other words, the predetermined number of clusters is enforced.

The third constraint enforces $c_{kl} = 1$ if clusters k and l are correlated: if x_{ik}, x_{jl}, d_{ij} are all equal to 1, the left hand side of the inequality is 1, and c_{kl} has to be 1 in order to satisfy the inequality. When x_{ik}, x_{jl} and d_{ij} are all equal to 1, it means component i is assigned to cluster k, while component j is assigned to cluster l, and i and j are correlated. In this case, clusters k and l are correlated as well, that is, c_{kl} should be one. If any of x_{ik}, x_{jl}, d_{ij} is not 1, c_{kl} could be either 0 or 1, but since we minimize

the sum of all c_{kl} in the objective (will be explained later in detail), c_{kl} automatically becomes 0 when clusters k and l are not correlated.

The fourth constraint makes $n_{ij} = 1$ if components i and j are assigned to different clusters. The analysis is similar to the one above: n_{ij} must be 1 to satisfy the inequality when x_{ik} and x_{jl} are both 1. $x_{ij} = 1, x_{kl} = 1$ and $k \neq l$ means that components i and j are assigned to different clusters. Since we try to minimize the sum of all n_{ij} in the objective as well, n_{ij} will be 0 if not both of x_{ik} and x_{jl} are 1.

The purpose of the last constraint is to make z_{ij} 1 if components i and j are in the same cluster and they are uncorrelated. In the inequality, when $x_{ik} = 1, x_{jk} = 1$ and $d_{ij} = 0$, z_{ij} should be 1, z_{ij} could be either 0 or 1 otherwise. $x_{ik} = 1, x_{jk} = 1$ and $d_{ij} = 0$ means that component i is assigned to cluster k, and component j is assigned to cluster k as well, but components i and j are uncorrelated. If either of the conditions above are not satisfied, z_{ij} will be 0, since the sum of all z_{ij} is minimized in the objective function.

In sum, the first two constraints establish the mapping between the components and clusters, while the last three constraints establish the corresponding relation between variables c_{jl}, n_{ik}, z_{ij} and x_{ij}, d_{ij}. The three variables c_{jl}, n_{ik} and z_{ij} play main roles in the objective function. Next, I will explain the meaning of the objective function.

The objective function is composed by three terms, each of which corresponds to one main objective.

The first term is a summation of all c_{ij}. Since $c_{ij} = 1$ if cluster i and j are correlated, this term represents twice of the total number of correlated cluster pairs when minimized. (It is twice of the total number because c_{ij} and c_{ji} are equal and both added to the objective value).

The second term is the total number of 1's out of the clusters, because $d_{ij}n_{ij}$ is nonzero only when $d_{ij} = 1$ and $n_{ij} = 1$, which means components i and j are correlated but they are not in the same cluster. Since the total number of 1's is a constant for a DSM, minimizing the total number of 1's out of the clusters is equivalent to maximizing the total number of 1's within the clusters.

The third term is the total number of the 0's that are within the clusters when minimized, since z_{ij} should be 1 if components i and j are uncorrelated but the entry (i, j) is in one of the clusters.

The weights w_1, w_2 and w_3 reflect the relative importance of the three objectives stated above. They can be adjusted to obtain different results.

Now, to show how well the model works, I illustrate the clustering results of the bicycle DSM shown in Figure 4-8. The weight for each objective is either 1 or 5, and all weight combinations are examined. More specifically, I use all the 7 weight combinations (1,1,1), (1,1,5), (1,5,1), (1,5,5), (5,1,1), (5,5,1) in the experiments.

First, we try to divide the components into three clusters. Figure 4-10 to Figure 4-13 show the clustering results. Figure 4-10 corresponds to the objective weights (w_1, w_2, w_3) = (1,1,1), (5,5,1), (1,5,1), Figure 4-11 corresponds to weights (w_1, w_2, w_3) = (1,1,5), (1,5,5), Figure 4-12 corresponds the weight (w_1, w_2, w_3) = (5,1,1), and Figure 4-12 corresponds weight (w_1, w_2, w_3) = (5,1,5). Table 4.10 summarizes the values of the three objectives (the number of correlated cluster pairs, the number of 1's out of the clusters, the number of 0's within the clusters) corresponding to the 4 clustering results.

Figure	Weight combination	Correlated pairs	1's outside	0's inside
4-10	(1,1,1), (5,5,1), (1,5,1)	4	4	10
4-11	(1,1,5), (1,5,5)	6	6	6
4-12	(5,1,1)	4	6	8
4-13	(5,1,5)	4	6	8

Table 4.10: The three objective values corresponding to the results shown in Figures 4-10 to 4-13. The three objectives are: 1) the number of correlated cluster pairs; 2) the number of 1's outside the clusters; 3) the number of 0's within the clusters.

From Table 4.10, we can see that the result shown in Figure 4-10 minimizes the number of correlated pairs and the number of 1's outside the clusters. However, the number of 0's wihtin the clusters is relatively large. This is because in the weight sets (1,1,1), (5,5,1), (1,5,1), the third objective is all weighted the least. From Figure 4-10, we can see that it clusters many components into one big cluster (the second cluster), since it includes more 1's into the clusters, and in the mean time reduces the

	Pedals	Chain	Gear Shift	Gears	Wheels	Handlebars	Odometer	Brake	Brake String
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1		1			
Gears		1	1	1	1				
Wheels				1	1		1	1	
Handlebars			1			1	1		
Odometer					1	1	1		
Brake					1			1	1
Brake String								1	1

Figure 4-10: Clustering result 1 of the bicycle DSM. Cluster components into 3 groups. Objective weights (1,1,1), (5,5,1), (1,5,1)

	Pedals	Chain	Gears	Wheels	Brake	Brake String	Gear Shift	Handlebars	Odometer
Pedals	1	1							
Chain	1	1	1						
Gears		1	1	1			1		
Wheels			1	1	1				1
Brake				1	1	1			
Brake String					1	1			
Gear Shift			1				1	1	
Handlebars							1	1	1
Odometer				1				1	1

Figure 4-11: Clustering result 2 of the bicycle DSM. Cluster components into 3 groups. Objective weights (1,1,5), (1,5,5).

interaction between the small clusters.

The objective values of the result shown in Figure 4-11 are more balanced. Correspondingly, the components are divided into 3 clusters with equal number of components. The disadvantage of this clustering, however, is that all clusters are correlated with each other. It is because that the weight for the first objective value is small, which means the number of correlated pairs is a relatively minor factor when these weights are applied in the clustering model.

Comparing to result 1 shown in Figure 4-10, the third and fourth results reduces the number of 0's inside the clusters, while sacrificing the number of 1's outside the clusters. This generates a more balanced clustering than result 1, although not as balanced as result 2.

After showing the 3-group clustering results, I now introduce the results of 4-group clustering. Figure 4-14 to Figure 4-17 shows the 4-group clustering results. Figure 4-14 corresponds to the objective weight $(w_1, w_2, w_3) = (1,1,1)$. Note that this clustering is exactly the same as the one shown in Figure 4-8 (the clustering adopted

	Pedals	Chain	Gear Shift	Gears	Brake	Brake String	Wheels	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1			1		
Brake					1	1	1		
Brake String					1	1			
Wheels				1	1		1		1
Handlebars			1					1	1
Odometer							1	1	1

Figure 4-12: Clustering result 3 of the bicycle DSM. Cluster components into 3 groups. Objective weight (5,1,1).

	Pedals	Chain	Gears	Wheels	Brake	Brake String	Gear Shift	Handlebars	Odometer
Pedals	1	1							
Chain	1	1	1						
Gears		1	1	1			1		
Wheels			1	1	1				1
Brake				1	1	1			
Brake String					1	1			
Gear Shift			1				1	1	
Handlebars							1	1	1
Odometer				1				1	1

Figure 4-13: Clustering result 4 of the bicycle DSM. Cluster components into 3 groups. Objective weight (5,1,5).

in [31]). Figure 4-15 corresponds to weights (w_1, w_2, w_3) = (5,1,5), (5,1,1), Figure 4-16 corresponds weights (w_1, w_2, w_3) = (5,1,1), (1,5,5), and Figure 4-17 corresponds weights (w_1, w_2, w_3) = (1,5,1), (5,5,1). Table 4.11 summarizes the values of the three objectives (the number of correlated cluster pairs, the number of 1's out of the clusters, the number of 0's within the clusters) corresponding to the 4 clustering results. Note that there are only two sets of objective values, which are 6,8,2 and 6,6,10, respectively. There are, however, several different clusterings corresponding to the objective values 6,8,2.

Figure	Weight combination	Correlated pairs	1's outside	0's inside
4-14	(1,1,1)	6	8	2
4-15	(5,1,5), (5,1,1)	6	8	2
4-16	(1,1,5), (1,5,5)	6	8	2
4-17	(1,5,1), (5,5,1)	6	6	10

Table 4.11: The three objective values corresponding to the results shown in Figures 4-14 to 4-17. The three objectives are: 1) the number of correlated cluster pairs; 2) the number of 1's outside the clusters; 3) the number of 0's within the clusters.

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

Figure 4-14: Clustering result 5 of the bicycle DSM. Cluster components into 4 groups. Objective weight (1,1,1).

	Pedals	Chain	Gears	Wheels	Brake	Brake String	Gear Shift	Handlebars	Odometer
Pedals	1	1							
Chain	1	1	1						
Gears		1	1	1			1		
Wheels			1	1	1				1
Brake				1	1	1			
Brake String					1	1			
Gear Shift			1				1	1	
Handlebars							1	1	1
Odometer				1				1	1

Figure 4-15: Clustering result 6 of the bicycle DSM. Cluster components into 4 groups. Objective weights (5,1,5), (5,1,1), (1,5,5).

	Pedals	Chain	Gears	Wheels	Odometer	Brake	Brake String	Gear Shift	Handlebar
Pedals	1	1							
Chain	1	1	1						
Gears		1	1	1				1	
Wheels			1	1	1	1			
Odometer				1	1				1
Brake				1		1	1		
Brake String						1	1		
Gear Shift			1					1	1
Handlebars					1			1	1

Figure 4-16: Clustering result 7 of the bicycle DSM. Cluster components into 4 groups. Objective weights (1,1,5).

	Pedals	Chain	Brake	Brake String	Gear Shift	Gears	Wheels	Handlebars	Odomete
Pedals	1	1							
Chain	1	1				1			
Brake			1	1			1		
Brake String			1	1					
Gear Shift					1	1		1	
Gears		1			1	1	1		
Wheels			1			1	1		1
Handlebars					1			1	1
Odometer							1	1	1

Figure 4-17: Clustering result 8 of the bicycle DSM. Cluster components into 4 groups. Objective weights (1,5,1), (5,5,1).

Camparing the 4-group clustering results to the 3-group clustering results, we can see that except the results corresponding to objective weights (1,5,1), (5,5,1) (shown in Figure 4-17), the number of 0's in the clusters is significantly reduced. It means that under 4-group clustering, the components in one cluster are more closely related, but on the other hand, the number of 1's outside the clusters and the number of correlated cluster pairs modestly increase, due to the increase of interfaces between clusters.

From the results above, we can see that the optimization model gives the decision makers the flexibility to generate different clusterings. From all the 8 possible clusterings (actually, there are more possible weight combinations that could generate other good clusterings), the decision maker can choose the one that best matches the practical needs.

In practice, the DSM model also works well. I apply the optimization model to solve the jet engine design problem demonstrated in [37]. The optimization model is solved by CPLEX (a mixed integer programming solver). The original problem and the corresponding clustering is shown in Figure 4-18. All the components are divided into 6 clusters.

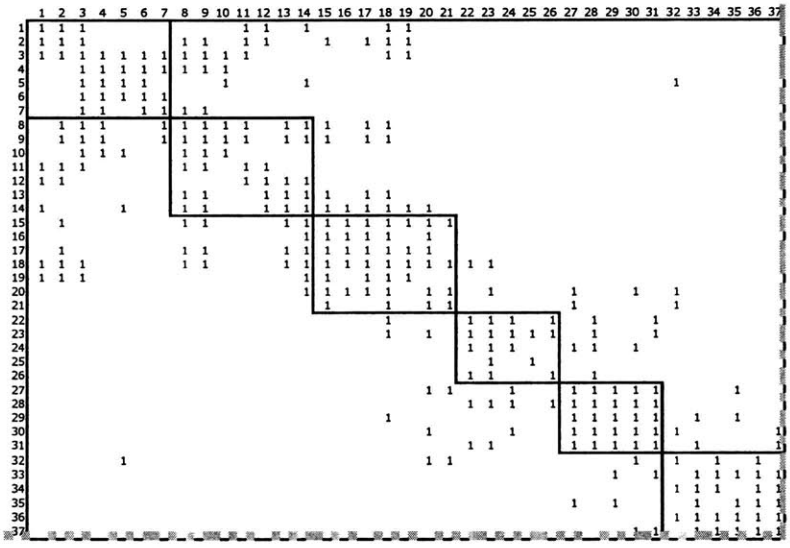


Figure 4-18: A clustering of the jet engine design problem provided by [37]

In order to compare the results, I also fix the number of clusters to be 6 (I will

introduce how to rid of the requirement of group number later in this section). I use four different weight sets. The first is (1, 1, 1), and the result is shown in Figure 4-19. (As a reminder, the three weights correspond to the number of correlated clusters, the 1's out of the clusters and the 0's in the clusters, respectively.) The weight sets of the second, third and fourth are (1, 2, 5), (1, 5, 5) and (5, 1, 2), respectively. The clustering results are shown in Figure 4-20, 4-21 and 4-22, respectively. The objective values of the four results are shown in Table 4.12.

Figure	Weight combination	Correlated pairs	1's outside	0's inside
4-18	Result in [37]	18	131	60
4-19	(1,1,1)	18	103	60
4-20	(1,2,5)	16	121	54
4-21	(1,5,5)	18	107	56
4-22	(5,1,2)	16	105	60

Table 4.12: The values of the three objectives

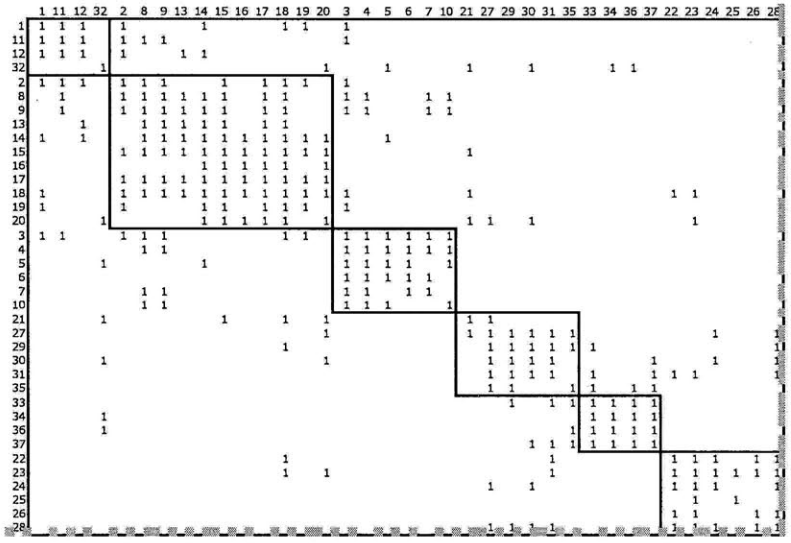


Figure 4-19: Result of solving the jet engine design problem, using weight set (1, 1, 1).

From Table 4.12, we can see that when the the three objectives are considered, the results obtained by the optimization model dominate the result shown in [37].

In most of the design problems, however, engineers do not know how many clusters should the components be divided into. They can always try different numbers, of

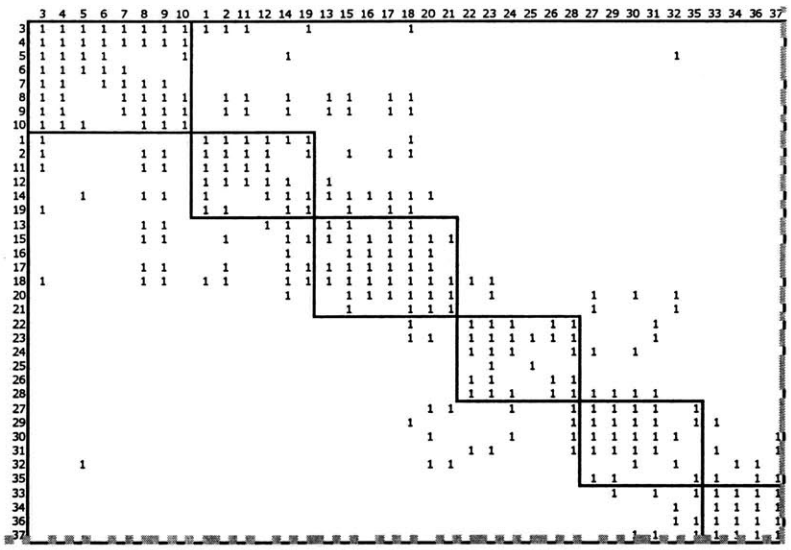


Figure 4-20: Result of solving the jet engine design problem, using weight set (1, 2, 5).

course, but it is a waste of time to solve the same problem repeatedly. Another issue is that when the number of components grows, it takes hours to divide them into several clusters.

Thus, I propose a method here so that one can solve large-scale clustering problems without prefixing the number of clusters, and in significantly shorter time. The idea is straightforward: first decompose the components into two clusters. Check the clustering result and the number of components in each cluster. If there are still too many components in one cluster, we decompose it into halves again. Keep doing the decomposition recursively until there are less than a score of components in each cluster. The clustering should be much easier now. One can try different numbers of clusters to find the best one. Since when the number of components is large, we only need to decompose the components into two clusters, the time needed to solve the large problems is significantly shortened. Also, since the number of components are reduced exponentially with the number of steps, the time needed to solve the problems is also shortened as the process goes. Since the method tries to divide the components into haves in the early stages, I call this approach the binary clustering method.

I apply the binary clustering method to the jet engine design problem. First,

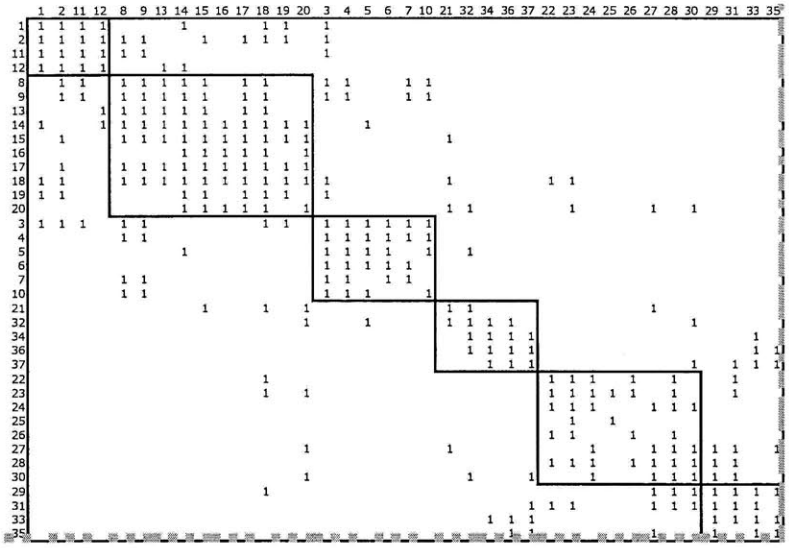


Figure 4-21: Result of solving the jet engine design problem, using weight set (1, 5, 5).

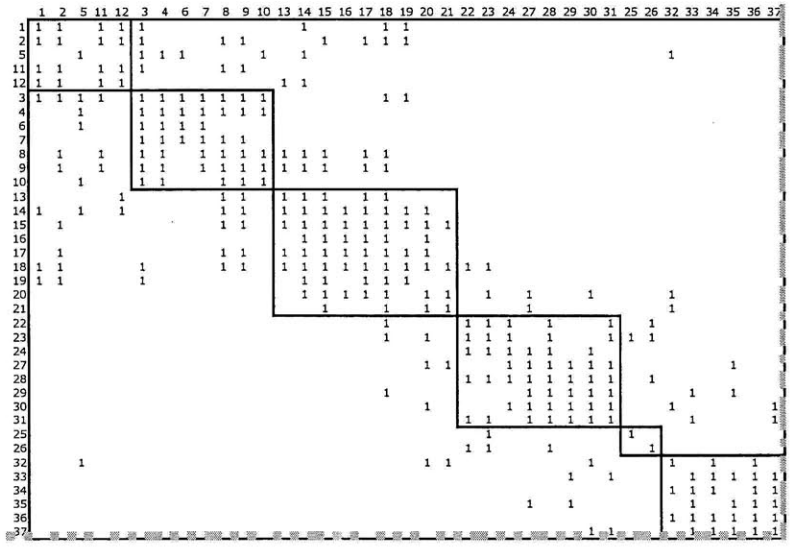


Figure 4-22: Result of solving the jet engine design problem, using weight set (5, 1, 2).

decompose it into two clusters. The result is shown in Figure 4-23. The first cluster has 19 components in total and the second has 18. In the next step, I decompose both of them into three clusters. The final clustering result is shown in Figure 4-24. The time of solving the original problem directly (divide the components into 6 clusters) is more than an hour. The time it takes of using the new approach is less than 20 minutes in total. The comparison of the objective values is shown in Table 4.13.

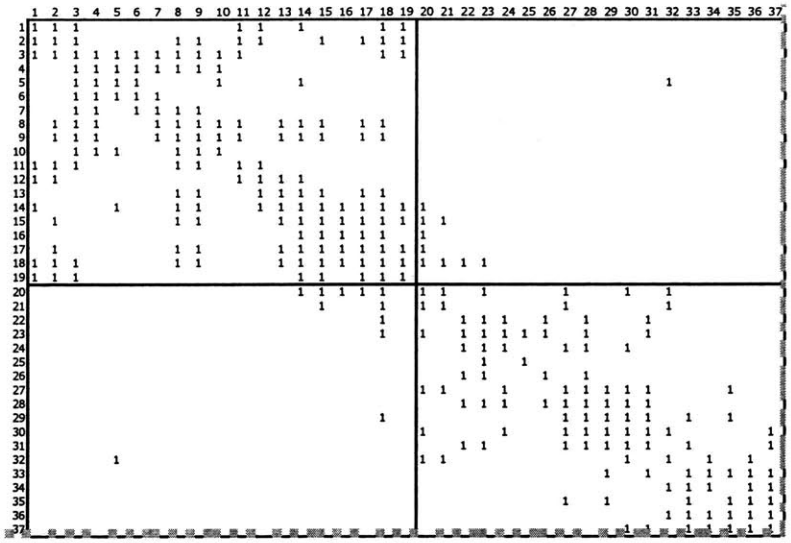


Figure 4-23: Applying the binary clustering method to the jet engine design problem, step 1

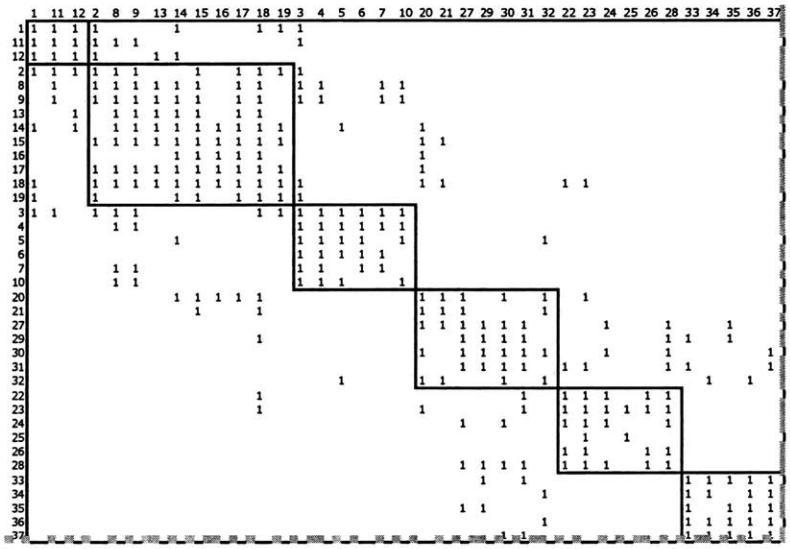


Figure 4-24: Applying the binary clustering method to the jet engine design problem, step 2

Although the clustering shown in Figure 4-24 could be further optimized, it is already a relatively good result, as shown in Table 4.13. Actually, it is not dominated by any of the other clusterings. Especially, it also dominates the result shown in [37]. When the number of components grows to fifty or more, the complete optimization model might not be solvable in limited time, but with the approach introduced above, one can still obtain a good clustering of a large-scale system.

Figure	Weight combination	Correlated pairs	1's outside	0's inside
4-18	Result in [37]	18	131	60
4-19	(1,1,1)	18	103	60
4-20	(1,2,5)	16	121	54
4-21	(1,5,5)	18	107	56
4-22	(5,1,2)	16	105	60
4-24	Binary clustering	16	109	56

Table 4.13: The values of the three objectives

4.3.3 Multiple DSM Clustering

In addition to clustering a single DSM, sometimes we need to cluster two or more DSMs at the same time. For example, when deciding how to divide the bicycle parts into several clusters, we might not only care about their assembly features, but also the dealer information of each parts. Figure 4-25 (the same as Figure 4-8) shows the assembly features, and Figure 4-26 shows the dealer information. To be specific, if two components can be purchased as a batch with a discount from a dealer, the corresponding entry in the DSM is marked by 1. (The dealer information is imaginary, just for illustration of the method).

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1	1				
Wheels				1	1	1			1
Brake					1	1	1		
Brake String						1	1		
Handlebars			1					1	1
Odometer					1			1	1

Figure 4-25: Bicycle assembly feature DSM

Our problem now becomes: how to cluster the components to achieve the all the goal stated in Section 4.3.1 for both the assembly DSM and the dealer DSM? If we view the optimization goals for the assembly DSM as one objective and the goals for the dealer DSM as another, we have a two-objective optimization problem. As in almost all multi-objective optimization problems, there exists trade-offs between the two objectives. One clustering scheme for assembly DSM might turn out to be a bad clustering for the dealer DSM, and vice versa. What I will provide is a way to find all

	Pedals	Chain	Gear Shift	Gears	Wheels	Brake	Brake String	Handlebars	Odometer
Pedals	1					1	1	1	
Chain		1	1		1			1	
Gear Shift		1	1		1				
Gears				1					1
Wheels		1	1		1	1		1	
Brake	1				1	1		1	
Brake String	1						1	1	
Handlebars	1	1			1	1	1	1	
Odometer				1					1

Figure 4-26: Dealer information DSM

the solutions on the convex Pareto front (refer to Section 2.1.1 for the definition of convex Pareto front), so that the decision maker can pick a most favorable clustering which is balanced between the two DSMs.

Mathematically, our optimization model becomes the one shown in Table 4.15. The notations are shown here again in Table 4.14.

x_{ik}	-	A binary decision variable indicating whether to assign component i to cluster k.
c_{kl}	-	A binary variable indicating whether clusters k and l are correlated.
n_{ij}	-	A binary variable indicating whether components i and j are in different clusters.
z_{ij}	-	A binary variable indicating whether components i and j are in the same cluster and the entry (i, j) in the DSM is 0.
d_{ij}	-	The value of the entry (i, j) in the DSM.
w_1	-	The weight of the number of correlated cluster pairs (one of the three main goals).
w_2	-	The weight of the number of 1's within clusters (one of the three main goals).
w_3	-	The weight of the number of 0's within clusters (one of the three main goals).
N	-	The number of components.
G	-	The number of clusters.

Table 4.14: The notations used in the integer programming model for DSM clustering.

Everything remains unchanged except the subscripts of c, d, n and z become three dimensional, with an extra one t representing the t'th DSM. I also assume that there are T DSMs to be clustered at the same time. Notice that there are T objective functions in total. In our bicycle example, $T = 2$, t=1 or 2, and we have a 2-objective optimization problem.

There are several ways of solving multi-objective optimization problems. In Sections 2.3 and 2.4, I introduced several efficient ones. The recursive knee algorithm mentioned in Section 2.4.1, in particular, also fits our need here.

$$\min \quad w_1 \cdot \sum_{i=1}^{G}\sum_{k=1}^{G} c_{ikt} + w_2 \cdot \sum_{i=1}^{N}\sum_{j=1}^{N} d_{ijt}n_{ijt} + w_3 \cdot \sum_{i=1}^{N}\sum_{j=1}^{N} z_{ijt}, \qquad \forall\, t = 1 \cdots T$$

$$\text{s.t.} \quad \sum_{k=1}^{G} x_{ik} = 1, \qquad \forall\, i = 1 \cdots N$$

$$\sum_{i=1}^{N} x_{ik} \geq 1, \qquad \forall\, k = 1 \cdots G$$

$$x_{ik} + x_{jl} + d_{ijt} - 2 \leq c_{klt}, \qquad \forall\, i, j = 1 \cdots N, \quad \forall\, k, l = 1 \cdots G, j \neq l, \quad \forall\, t = 1 \cdots T$$

$$x_{ik} + x_{jl} - 1 \leq n_{ijt}, \qquad \forall\, i, j = 1 \cdots N, \quad \forall\, k, l = 1 \cdots G, k \neq l, \quad \forall\, t = 1 \cdots T$$

$$x_{ik} + x_{jk} + (1 - d_{ijt}) - 2 \leq z_{ijt}, \qquad \forall\, i, j = 1 \cdots N, \quad \forall\, k = 1 \cdots G, \quad \forall\, t = 1 \cdots T$$

Table 4.15: Integer programming model for the multiple DSM clustering problem.

Here I will briefly go through the recursive knee algorithm again. More detailed discussion could be found in Section 2.4.1.

As shown in Figure 4-27, the recursive knee algorithm starts with optimizing over each single objective. In the 2-dimensional example, the algorithm optimizes the clustering of each of the two DSMs respectively. The two optimal clusterings correspond to points 1 and 2. Then the algorithm calculates the gradient of the line connecting points 1 and 2 and try to push it in the orthogonal direction as far as possible. Mathematically, the algorithm computes the weights that should be applied to each DSM and optimize the weighted sum of the objectives with the original constraints. By solving the new problem, point 3 will be found. The recursive process then repeat itself between points 1, 3 and 2, 3. Such process stops when no more points that are further in the orthogonal direction can be found.

Using the optimization algorithm above, we can obtain the convex Pareto front for the T-objective optimization problem. Figure 4-28 shows the convex Pareto front (the weights are $w_1 = 1, w_2 = 2, w_3 = 5$).

The X axis represents the objective value of the assembly DSM, and the Y axis

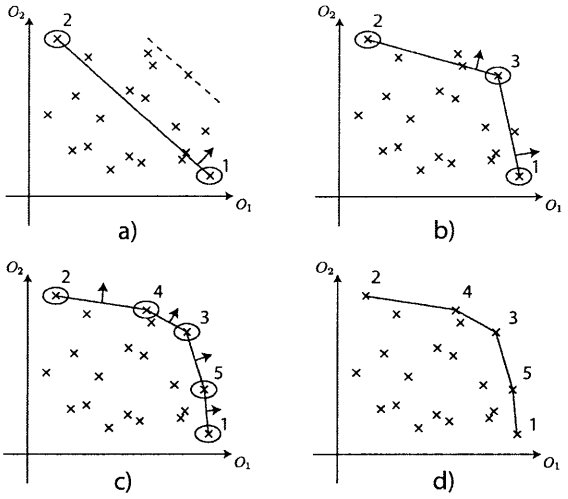


Figure 4-27: An illustration of the recursive knee algorithm. The detailed description of the algorithm can be found in Section 2.4.1.

represents the objective value of the dealer DSM. The numerical values do not have practical meanings, but the relative positions of the points convey some information: Point 1 optimizes the objective value for the assembly DSM, but does not achieve a satisfactory result for the dealer DSM. It means that the corresponding clustering provides a good division for the assembly DSM, but is not a good one for the dealer DSM. Point 2, on the other hand, is a good clustering for the dealer DSM, but not a good one for the assembly DSM. Point 3, 4 and 5 provide some trade-off between the two extreme points.

One step further, let's look into the clusterings corresponding to the 5 points. As what we can see in Figure 4-29, the clustering for the first DSM (the assembly DSM) is well clustered, but the second (the dealer DSM) is not. This clustering corresponds to point 1, which optimizes the assembly DSM clustering, but is not a good choice for the dealer DSM clustering.

In Figure 4-30, on the other hand, the clustering for the first DSM (the assembly DSM) is not so well clustered, but the second DSM (the dealer DSM) is very well clustered. This clustering corresponds to point 2, which optimizes the dealer DSM clustering, but not a good solution for the assembly DSM clustering.

Point 3 gives the decision makers a possible trade-off. Its clustering is shown in Figure 4-31. As we can see in the figure, neither of the clustering for the first nor

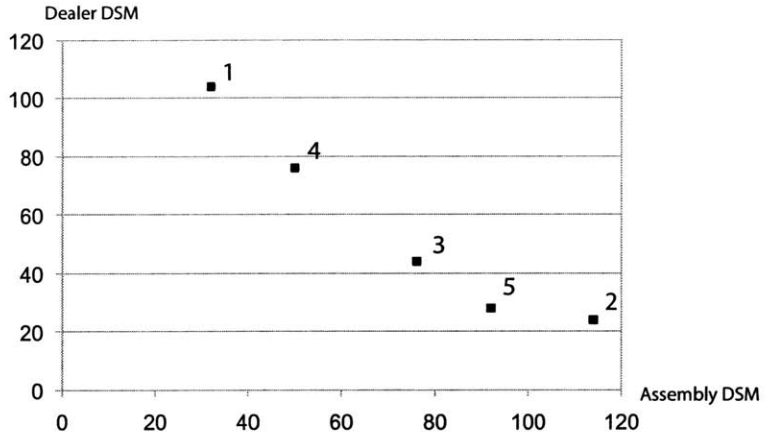


Figure 4-28: Plot of the convex Pareto front of the multiple bicycle DSM clustering problem.

	Pedals	Chain	Gear Shift	Gears	Brake	Brake String	Wheels	Handlebars	Odometer
Pedals	1	1							
Chain	1	1		1					
Gear Shift			1	1				1	
Gears		1	1	1			1		
Brake					1	1	1		
Brake String					1	1			
Wheels				1	1		1		1
Handlebars			1					1	1
Odometer							1	1	1

	Pedals	Chain	Gear Shift	Gears	Brake	Brake String	Wheels	Handlebars	Odometer
Pedals	1				1	1		1	
Chain		1	1				1	1	
Gear Shift		1	1				1		
Gears				1					1
Brake	1				1		1	1	
Brake String	1					1		1	
Wheels		1	1		1		1	1	
Handlebars	1	1			1	1	1	1	
Odometer				1					1

Figure 4-29: The multiple bicycle DSM clustering result corresponding to point 1 in Figure 4-28.

the second is optimized, but both of the two clusterings are somehow better than the worst cases in the previous two clusterings.

Finally, let's look at the clustering corresponding to point 4, which is shown in Figure 4-32. This clustering might be the best trade-off for decision makers. It is not a perfect clustering for either of the DSMs, but is an acceptable clustering for both of them.

	Pedals	Brake	Handlebars	Chain	Gear Shift	Wheels	Brake String	Gears	Odometer
Pedals	1			1					
Brake		1				1	1		
Handlebars			1		1				1
Chain	1			1				1	
Gear Shift			1		1			1	
Wheels		1				1		1	1
Brake String		1					1		
Gears				1	1	1		1	
Odometer			1			1			1

	Pedals	Brake	Handlebars	Chain	Gear Shift	Wheels	Brake String	Gears	Odometer
Pedals	1	1	1				1		
Brake	1	1	1			1			
Handlebars	1	1	1	1		1	1		
Chain			1	1	1	1			
Gear Shift				1	1	1			
Wheels		1	1	1	1	1			
Brake String	1		1				1		
Gears								1	1
Odometer								1	1

Figure 4-30: The multiple bicycle DSM clustering result corresponding to point 2 in Figure 4-28.

	Pedals	Brake String	Chain	Gear Shift	Wheels	Brake	Handlebars	Gears	Odometer
Pedals	1		1						
Brake String		1				1			
Chain	1		1					1	
Gear Shift				1			1	1	
Wheels					1	1		1	1
Brake		1			1	1			
Handlebars				1			1		1
Gears			1	1	1			1	
Odometer					1		1		1

	Pedals	Brake String	Chain	Gear Shift	Wheels	Brake	Handlebars	Gears	Odometer
Pedals	1	1				1	1		
Brake String	1	1					1		
Chain			1	1	1		1		
Gear Shift			1	1	1				
Wheels			1	1	1	1	1		
Brake	1				1	1	1		
Handlebars	1	1	1		1	1	1		
Gears								1	1
Odometer								1	1

Figure 4-31: The multiple bicycle DSM clustering result corresponding to point 3 in Figure 4-28.

4.3.4 Summary

In this section, I introduced the DSM clustering problem and gave specific mathematical definitions of some critical goals that should be considered in the clustering

	Pedals	Brake String	Chain	Gear Shift	Gears	Wheels	Brake	Handlebars	Odometer
Pedals	1		1						
Brake String		1					1		
Chain	1		1		1				
Gear Shift				1	1			1	
Gears			1	1	1	1			
Wheels					1	1	1		1
Brake		1				1	1		
Handlebars				1				1	1
Odometer						1		1	1

	Pedals	Brake String	Chain	Gear Shift	Gears	Wheels	Brake	Handlebars	Odometer
Pedals	1	1					1	1	
Brake String	1	1						1	
Chain			1	1		1		1	
Gear Shift			1	1		1			
Gears					1				1
Wheels			1	1		1	1	1	
Brake	1					1	1	1	
Handlebars	1	1	1			1	1	1	
Odometer					1				1

Figure 4-32: The multiple bicycle DSM clustering result corresponding to point 4 in Figure 4-28

problems. I then provided an integer programing model based on these goals to solve the problem. The model is very flexible and could provide the decision makers different clustering schemes by allowing them to adjust the weights of the different goals.

I also introduced the new multiple DSM clustering problem and demonstrated the algorithm that solves the problem. The algorithm is a combination of the integer programming model introduced in Section 4.3.2 and the recursive knee algorithm introduced in Section 2.4.1.

In each of the subsection mentioned above, I provided some examples. The results of the examples show that the optimization model is effective for solving the DSM clustering problems. Especially, the result of the jet design problem shows that the model could be used to solve practical problems.

The optimization model provides a new method for clustering DSMs. There are some other popular methods that could solve the clustering DSMs as well. Refer to [5], [45], [31], [29] for details. In the analysis in Section 4.3.2, we can see that the quality of the clustering obtained by solving the optimization model introduced here

is comparable to, if not better than, the existing methods.

4.4 Summary

In this chapter, I mainly discuss two problems: one is how to bridge the graph representation of systems and the matrix representation of systems. The other is how to effectively cluster the DSMs. For the first problem, after extending the classical DSM, I reveal the Markov property of the extended DSM and provide examples showing the connection between the graphical representation of systems and the extended DSM. Then I explain the concept of projection and show the practical use of projection on system graphs as well as DSMs. Moreover, I further exploit the bi-partite property of OPDs, and show that such a special property can be used to further condense the corresponding DSM and leads to better computational efficiency. I end the discussion with two examples projecting an OPD to objects only and processes only.

For the DSM clustering, I first clarify the goals we want to achieve and describe them in formal mathematical form. Secondly, I use an integer programming model to express the problem. The optimization model leaves the weights on each objective as parameters and allow the system architects to adjust the combination of values to attain different results. In addition to the pedagogical example of bicycle parts, I provide an example of jet engine design, showing that the optimization model is practically effective. After the demonstration of the single DSM clustering, I introduce the multiple DSM clustering problem and use the recursive knee algorithm illustrated in section 2.4.1 to solve it. I apply the method to the extended bicycle parts problem and explain the result.

Chapter 5

Contributions and Future Work

5.1 Contributions

In this thesis, I address three problems in two domains: decision making domain and system architecting domain. The first problem, which is in the decision-making domain, is about how to formulate and solve the decision-making problems in complex/large-scale systems. In Chapter 2, I start with explaining what is Constraint Optimization Problem (COP) and showing that many decision-making problems have the same fundamental parts as COPs. Then I demonstrate how to model the decision-making problems with COPs. Next, to address the issue of solving multi-objective COPs, I first illustrate an efficient algorithm, the Conflict-directed A* (CDA*), that can solve the single-objective COPs efficiently. I then combine the recursive knee approach and the CDA* algorithm to develop a new algorithm that can be used to solve the multi-objective COPs. The new algorithm is further extended to find solutions that are in a controllable region close to the convex Pareto front. Finally, I propose an improved algorithm which incorporates a feature in the Opportunistic Improvement Algorithm (OIA) that can help find the complete Pareto front.

To further cement the claim that many decision-making problems can be modelled and solved as COPs, I provide two case studies of practical problems. One is the Apollo mission design problem, and another is the NASA decadal survey problem. From the study, we can see that the COP can well encode the practical decision-

making problems, and the results obtained by solving the COP models provide good insights for decision makers.

The second problem, which is in the system architecture domain, is how to translate between the graphical representation of systems and the matrix representation of systems. To solve this problem, I first extend the classical DSM to make it possible to store various information of the links between components in a system. After the extension of DSM, I reveal the Markov property of the extended DSM, and show how to use the DSM as a computational tool for the system graphs. Furthermore, I define the concept of projection that maps and condenses system graphs to the DSM. The projection operation can effectively condense the information of the graphical representation of systems, and emphasize the parts that are of interest to the system architects. In addition, I make use of the bi-partite property of the OPDs to further improve the efficiency of the projection from OPD to DSM.

The last problem, which is also in the system architecture domain, is about how to cluster the DSMs. I formally define the goal of the clustering problem and develop an optimization model which captures the critical features of the problem. I use the jet engine design to show that the optimization model effectively solves the clustering problems while leaving much flexibility to the system architectures, who can adjust the parameters of the model to obtain various results. Moreover, I introduce a new multiple DSM clustering problem, and extend the optimization model for the single DSM clustering problem to solve the multiple DSM clustering problem.

In sum, the main contributions in this thesis are:

- Develop an approach to model decision-making problems as multi-objective COPS.
- Develop a new algorithm that finds the convex Pareto front of the multi-objective COP.
- Extend the previous algorithm to obtain the solutions that are close to the convex Pareto front.
- Propose an extension of the algorithm to find the complete Pareto front.

- Extend the classical DSMs and reveal its Markov property.
- Introduce the concept of projection which connects the graphical representation of systems and the matrix representation of systems.
- Formalize the DSM clustering problems and provide an optimization model that encodes the problems.
- Introduce multiple DSM clustering problems and extend the previous optimization model to solve the problems.
- Provide applications that test all the algorithms, methods and models above.

5.2 Future Work

For each of the three main problems that I discuss in this thesis, there are still some more important works needed to be done. For the expression of the COPs, we need to develop a more efficient and compact language and the corresponding compiler which can describe the practical problems efficiently and translate the descriptions of the practical problems to large-scale COPs. For example, in the case study of the decadal survey, when we need to express $x_i \neq x_j, \quad i \neq j$, we need to expand it to (NOT $x_i \neq 2010$) OR (NOT $x_j \neq 2010$), (NOT $x_i \neq 2011$) OR (NOT $x_j \neq 2011$), $\cdots$, (NOT $x_i \neq 2020$) OR (NOT $x_j \neq 2020$). This becomes even more cumbersome when we want to prolong the time span to 20 years. Thus, in order to express more practical problems and to express them more efficiently, we need a better language. For the problem solving part, we need to implement the improved algorithm proposed in section 2.4.3, and we also need some benchmark problems other than practical problems to test the efficiency of the new algorithms.

For the second problem, i.e., the projection from system graphs to DSM, we might be able to further exploit the eigen space of the symbolic DSM, as mentioned in section 4.2.4. The computational efficiency of the projection and the calculation of the transition relationships between components in the systems would be significantly improved if the structure of the eigen space of DSM is proved to be special.

For the DSM clustering problem, although treating the weights of different objectives as parameters provide the system architects much flexibility, sometimes they need guidance to choose the parameters. An interesting future work might be researching how to optimize the parameters according to people's needs. Another possible improvement of the model is to cancel the fixed number of clusters and make it the fourth objective, i.e., to maximize the number of clusters while keeping the original three objectives. If this could be done, we save the system architects the trouble of choosing the right number of clusters.

Appendix A

The Multi-objective Constraint Optimization Model for Apollo Mission Mode Study

```
SYSTEM MODEL NAMED (MPL::VALVE-MODULE MPL::MY-VAVE)

VARIABLE DOMAIN TYPE DEFINITIONS

  VARIABLE-DOMAIN-TYPE MPL::CM-CREW-SIZE
                       (MPL::CM-2 MPL::CM-3)
  VARIABLE-DOMAIN-TYPE MPL::LM-CREW-SIZE
                       (MPL::LM-0 MPL::LM-1 MPL::LM-2 MPL::LM-3)
  VARIABLE-DOMAIN-TYPE MPL::LM-FUEL-TYPE
                       (MPL::LM-CRYOGENIC MPL::LM-STORABLE MPL::LM-NA)
  VARIABLE-DOMAIN-TYPE MPL::SM-FUEL-TYPE
                       (MPL::SM-CRYOGENIC MPL::SM-STORABLE)
  VARIABLE-DOMAIN-TYPE MPL::BINARY-DECISION
                       (MPL::YES MPL::NO)
  VARIABLE-DOMAIN-TYPE MPL::ORBIT-OR-DIRECT
                       (MPL::ORBIT MPL::DIRECT)
```

```
VARIABLE DEFINITIONS

  INSTANCE (MPL::APOLLO-MISSION MPL::APOLLO13)
    VARIABLE (MPL::MOON-DEPARTURE MPL::APOLLO13)
             DOMAIN-TYPE MPL::ORBIT-OR-DIRECT
    VARIABLE (MPL::MOON-ARRIVAL MPL::APOLLO13)
             DOMAIN-TYPE MPL::ORBIT-OR-DIRECT
    VARIABLE (MPL::EARTH-LAUNCH MPL::APOLLO13)
             DOMAIN-TYPE MPL::ORBIT-OR-DIRECT
    VARIABLE (MPL::EOR MPL::APOLLO13)
             DOMAIN-TYPE MPL::BINARY-DECISION
    VARIABLE (MPL::SM-FUEL MPL::APOLLO13)
             DOMAIN-TYPE MPL::SM-FUEL-TYPE
    VARIABLE (MPL::LM-FUEL MPL::APOLLO13)
             DOMAIN-TYPE MPL::LM-FUEL-TYPE
    VARIABLE (MPL::LM-CREW MPL::APOLLO13)
             DOMAIN-TYPE MPL::LM-CREW-SIZE
    VARIABLE (MPL::CM-CREW MPL::APOLLO13)
             DOMAIN-TYPE MPL::CM-CREW-SIZE
    VARIABLE (MPL::LOR MPL::APOLLO13)
             DOMAIN-TYPE MPL::BINARY-DECISION

STATE CONSTRAINT DEFINITIONS

  INSTANCE (MPL::APOLLO-MISSION MPL::APOLLO13)
    STATIC-CONSTRAINT (AND (OR (= (MPL::EARTH-LAUNCH MPL::APOLLO13) MPL::ORBIT)
                               (= (MPL::EOR MPL::APOLLO13) MPL::NO))
                           (OR (= (MPL::MOON-ARRIVAL MPL::APOLLO13) MPL::ORBIT)
                               (= (MPL::LOR MPL::APOLLO13) MPL::NO))
```

```
                        (OR (= (MPL::CM-CREW MPL::APOLLO13) MPL::CM-3)
                            (NOT (= (MPL::LM-CREW MPL::APOLLO13) MPL::LM-3)))
                        (OR (= (MPL::MOON-DEPARTURE MPL::APOLLO13) MPL::ORBIT)
                            (= (MPL::LOR MPL::APOLLO13) MPL::NO))
                        (OR (NOT (= (MPL::LM-CREW MPL::APOLLO13) MPL::LM-0))
                            (NOT (= (MPL::LOR MPL::APOLLO13) MPL::YES)))
                        (OR (= (MPL::LM-CREW MPL::APOLLO13) MPL::LM-0)
                            (= (MPL::LOR MPL::APOLLO13) MPL::YES))
                        (OR (NOT (= (MPL::LM-FUEL MPL::APOLLO13) MPL::LM-NA))
                            (NOT (= (MPL::LOR MPL::APOLLO13) MPL::YES)))
                        (OR (= (MPL::LM-FUEL MPL::APOLLO13) MPL::LM-NA)
                            (= (MPL::LOR MPL::APOLLO13) MPL::YES)))

UTILITY FUNCTION DEFINITIONS

  INSTANCE (MPL::APOLLO-MISSION MPL::APOLLO13)

    CRITERIA (MPL::IMLEO MPL::APOLLO13)
      PREFERENCE MIN
      COMPOSITION-OP #'+
      UTILITY-RELATION
        SCOPE ((MPL::SM-FUEL MPL::APOLLO13)
               (MPL::LOR     MPL::APOLLO13)
               (MPL::LM-FUEL MPL::APOLLO13)
               (MPL::LM-CREW MPL::APOLLO13))
        VALUED-TUPLES
              (((NIL MPL::NO NIL NIL) 0)
               ((NIL NIL NIL MPL::LM-0) 0)
               ((MPL::SM-CRYOGENIC MPL::YES MPL::LM-CRYOGENIC MPL::LM-1)
                5540.534)
```

```
((MPL::SM-STORABLE  MPL::YES MPL::LM-CRYOGENIC MPL::LM-1)
 7953.0024)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-STORABLE MPL::LM-1)
 9737.675)
((MPL::SM-STORABLE  MPL::YES MPL::LM-STORABLE MPL::LM-1)
 13977.67)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-NA MPL::LM-1)
 9737.675)
((MPL::SM-STORABLE  MPL::YES MPL::LM-NA MPL::LM-1)
 13977.67)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-CRYOGENIC MPL::LM-2)
 7387.3784)
((MPL::SM-STORABLE  MPL::YES MPL::LM-CRYOGENIC MPL::LM-2)
 10604.004)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-STORABLE MPL::LM-2)
 12983.566)
((MPL::SM-STORABLE  MPL::YES MPL::LM-STORABLE MPL::LM-2)
 18636.895)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-NA MPL::LM-2)
 12983.566)
((MPL::SM-STORABLE  MPL::YES MPL::LM-NA MPL::LM-2)
 18636.895)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-CRYOGENIC MPL::LM-3)
 9234.223)
((MPL::SM-STORABLE  MPL::YES MPL::LM-CRYOGENIC MPL::LM-3 )
 13255.004)
((MPL::SM-CRYOGENIC MPL::YES MPL::LM-STORABLE MPL::LM-3)
 16229.458)
((MPL::SM-STORABLE  MPL::YES MPL::LM-STORABLE MPL::LM-3)
 23296.117)
```

```
        ((MPL::SM-CRYOGENIC MPL::YES MPL::LM-NA MPL::LM-3)
         16229.458)
        ((MPL::SM-STORABLE  MPL::YES MPL::LM-NA MPL::LM-3)
         23296.117))

UTILITY-RELATION
  SCOPE ((MPL::SM-FUEL MPL::APOLLO13) (MPL::LM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((NIL                MPL::LM-0) 0)
                 ((MPL::SM-CRYOGENIC MPL::LM-1) 2612.4048)
                 ((MPL::SM-STORABLE  MPL::LM-1) 3749.9023)
                 ((MPL::SM-CRYOGENIC MPL::LM-2) 3483.206)
                 ((MPL::SM-STORABLE  MPL::LM-2) 4999.8696)
                 ((MPL::SM-CRYOGENIC MPL::LM-3) 4354.008)
                 ((MPL::SM-STORABLE  MPL::LM-3) 6249.8374))

UTILITY-RELATION
  SCOPE ((MPL::SM-FUEL MPL::APOLLO13)
         (MPL::LOR     MPL::APOLLO13)
         (MPL::CM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((NIL                MPL::YES NIL) 0)
                 ((MPL::SM-CRYOGENIC MPL::NO  MPL::CM-2) 78119.43)
                 ((MPL::SM-STORABLE  MPL::NO  MPL::CM-2) 168699.5)
                 ((MPL::SM-CRYOGENIC MPL::NO  MPL::CM-3) 107414.22)
                 ((MPL::SM-STORABLE  MPL::NO  MPL::CM-3) 231961.78))

UTILITY-RELATION
  SCOPE ((MPL::SM-FUEL MPL::APOLLO13) (MPL::CM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::SM-CRYOGENIC MPL::CM-2) 15880.687)
                 ((MPL::SM-STORABLE  MPL::CM-2) 24204.219)
                 ((MPL::SM-CRYOGENIC MPL::CM-3) 21835.945)
```

```
                ((MPL::SM-STORABLE  MPL::CM-3) 33280.797))

UTILITY-RELATION
  SCOPE ((MPL::LOR MPL::APOLLO13)
         (MPL::LM-FUEL MPL::APOLLO13)
         (MPL::LM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::NO NIL NIL) 0)
                 ((MPL::YES MPL::LM-CRYOGENIC MPL::LM-1) 15690.348)
                 ((MPL::YES MPL::LM-STORABLE  MPL::LM-1) 27576.312)
                 ((MPL::YES MPL::LM-NA        MPL::LM-1) 27576.312)
                 ((MPL::YES MPL::LM-CRYOGENIC MPL::LM-2) 20920.465)
                 ((MPL::YES MPL::LM-STORABLE  MPL::LM-2) 36768.42)
                 ((MPL::YES MPL::LM-NA        MPL::LM-2) 36768.42)
                 ((MPL::YES MPL::LM-CRYOGENIC MPL::LM-3) 26150.578)
                 ((MPL::YES MPL::LM-STORABLE  MPL::LM-3) 45960.523)
                 ((MPL::YES MPL::LM-NA        MPL::LM-3) 45960.523))

UTILITY-RELATION
  SCOPE ((MPL::LM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::LM-0) 0)
                 ((MPL::LM-1) 7398.1206)
                 ((MPL::LM-2) 9864.16)
                 ((MPL::LM-3) 12330.201))

UTILITY-RELATION
  SCOPE ((MPL::CM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::CM-2) 19728.32)
                 ((MPL::CM-3) 27126.441))
```

```
CRITERIA (MPL::RISK MPL::APOLLO13)
  PREFERENCE MAX
  COMPOSITION-OP #'*
  UTILITY-RELATION
    SCOPE ((MPL::EOR MPL::APOLLO13))
    VALUED-TUPLES (((MPL::YES) 0.9025)
                   ((MPL::NO)  0.98))

  UTILITY-RELATION
    SCOPE ((MPL::EARTH-LAUNCH MPL::APOLLO13))
    VALUED-TUPLES (((MPL::DIRECT) 0.90)
                   ((MPL::ORBIT)  0.99))

  UTILITY-RELATION
    SCOPE ((MPL::LOR MPL::APOLLO13))
    VALUED-TUPLES (((MPL::YES) 0.95)
                   ((MPL::NO)  1))

  UTILITY-RELATION
    SCOPE ((MPL::MOON-ARRIVAL MPL::APOLLO13))
    VALUED-TUPLES (((MPL::DIRECT) 0.90)
                   ((MPL::ORBIT)  0.9405))

  UTILITY-RELATION
    SCOPE ((MPL::MOON-DEPARTURE MPL::APOLLO13))
    VALUED-TUPLES (((MPL::DIRECT) 0.855)
                   ((MPL::ORBIT)  0.855))

  UTILITY-RELATION
```

```
  SCOPE ((MPL::CM-CREW MPL::APOLLO13)
         (MPL::LM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::CM-2 MPL::LM-2) 0.90)
 ((MPL::CM-3 MPL::LM-2) 1.0)
 ((MPL::CM-2 MPL::LM-3) 1.0)
 ((MPL::CM-3 MPL::LM-3) 0.90)
 ((NIL       MPL::LM-0) 1.0)
 ((NIL       MPL::LM-1) 1.0))

UTILITY-RELATION
  SCOPE ((MPL::LM-CREW MPL::APOLLO13))
  VALUED-TUPLES (((MPL::LM-1) 0.90) ((MPL::LM-0) 1.0)
 ((MPL::LM-2) 1.0)  ((MPL::LM-3) 1.0))

UTILITY-RELATION
  SCOPE ((MPL::SM-FUEL MPL::APOLLO13) (MPL::LOR MPL::APOLLO13))
  VALUED-TUPLES (((MPL::SM-CRYOGENIC MPL::YES) 0.9025)
         ((MPL::SM-CRYOGENIC MPL::NO)  0.8145)
 ((MPL::SM-STORABLE  NIL)      1.0))

UTILITY-RELATION
  SCOPE ((MPL::LM-FUEL MPL::APOLLO13))
  VALUED-TUPLES (((MPL::LM-CRYOGENIC) 0.9025)
 ((MPL::LM-STORABLE)  1.0))
```

Bibliography

[1] JM Borwein. The geometry of Pareto efficiency over cones. *Optimization*, 11(2):235–248, 1980.

[2] TR Browning, L.M.A. Co, and F. Worth. Applying the design structure matrix to system decomposition andintegration problems: a review and new directions. *IEEE Transactions on Engineering management*, 48(3):292–306, 2001.

[3] M. Carrascosa, S.D. Eppinger, and D.E. Whitney. Using the design structure matrix to estimate product development time. In *Proceedings of ASME Design Engineering Technical Conference*, 1981.

[4] V. Chankong and Y.Y. Haimes. *Multiobjective decision making: theory and methodology*. North-Holland Amsterdam, 1983.

[5] L. Chen and S. Li. Analysis of decomposability and complexity for design problems in the context of decomposition. *Journal of Mechanical Design*, 127:545, 2005.

[6] V.A. Chobotov. *Orbital mechanics*. AIAA, 2002.

[7] R.T. Clemen and T. Reilly. *Making hard decisions: an introduction to decision analysis*. Duxbury Press Belmont, CA, 1996.

[8] C.A.C. Coello, S. de Computacion, and C.S.P. Zacatenco. Twenty years of evolutionary multi-objective optimization: A historical view of the field. *IEEE Computational Intelligence Magazine*, 1(1), 2006.

[9] Justin M. Colson. System architecting of a campaign of earth observing satellites. Master's thesis, MIT, 2008.

[10] Edward F. Crawley. Esd.34 systems architecting. *Lecture notes, MIT Engineering Systems Division*, IAP 2007.

[11] I. Das. A preference ordering among various Pareto optimal alternatives. *Structural and Multidisciplinary Optimization*, 18(1):30–35, 1999.

[12] I. Das. On characterizing the "knee" of the Pareto curve based on normal-boundary intersection. *Structural and Multidisciplinary Optimization*, 18(2):107–115, 1999.

[13] R. Dechter. *Constraint processing.* Morgan Kaufmann, 2003.

[14] R. Dechter and J. Pearl. Generalized best-first search strategies and the optimality of A*. *Journal of the ACM (JACM)*, 32(3):505–536, 1985.

[15] D. Dori. *Object-process methodology: A holistic systems paradigm.* Springer, 2002.

[16] J.S. Dyer, P.C. Fishburn, R.E. Steuer, J. Wallenius, and S. Zionts. Multiple criteria decision making, multiattribute utility theory: the next ten years. *Management Science*, pages 645–654, 1992.

[17] P. Efficiency. The Theory of Vector Maximization,". *Journal of Mathematical Analysis and Applications*, 22(3):618–630, 1968.

[18] M. Ehrgott. Multiobjective Optimization. *AI Magazine*, 29(4):47, 2009.

[19] SD Eppinger. Innovation at the speed of information. *Harv Bus Rev*, 79(1):149–58, 2001.

[20] J. Figueira, S. Greco, and M. Ehrgott. *Multiple criteria decision analysis: state of the art surveys.* Springer Verlag, 2005.

[21] C.M. Fonseca, P.J. Fleming, et al. Genetic algorithms for multiobjective optimization: Formulation, discussion and generalization. In *Proceedings of the fifth international conference on genetic algorithms*, volume 423. Citeseer, 1993.

[22] R.G. Gallager. *Discrete stochastic processes.* Springer, 1996.

[23] D.E. Goldberg and J. Richardson. Genetic algorithms with sharing for multimodal function optimization. In *Proceedings of the Second International Conference on Genetic Algorithms on Genetic algorithms and their application table of contents*, pages 41–49. L. Erlbaum Associates Inc. Hillsdale, NJ, USA, 1987.

[24] J. Horn, N. Nafpliotis, and DE Goldberg. A niched Pareto genetic algorithm for multiobjective optimization. In *Evolutionary Computation, 1994. IEEE World Congress on Computational Intelligence., Proceedings of the First IEEE Conference on Genetic Algorithm*, pages 82–87, 1994.

[25] J.C. Houbolt, J.D. Bird, and M.J. Queijo. Guidance and Navigation Aspects of Space Rendezvous. *Control, Guidance, and Navigation of Spacecraft. NASA SP-17, pages, published by NASA, Washington, DC, December 1962, p. 15*, 1962.

[26] U. Lindemann, M. Maurer, and T. Braun. *Structural Complexity Management: An Approach for the Field of Product Design.* Springer Verlag, 2008.

[27] M. Mitchell. *An introduction to genetic algorithms.* The MIT press, 1998.

[28] J. Philip. Algorithms for the vector maximization problem. *Mathematical Programming*, 2(1):207–229, 1972.

[29] T.U. Pimmler and S.D. Eppinger. Integration analysis of product decompositions. In *Proceedings of the ASME Design Theory and Methodology Conference*, volume 68, pages 343–351, 1994.

[30] Derek Rayside, H.-Christian Estler, and Daniel Jackson. A Guided Improvement Algorithm for Exact, General Purpose, Many-Objective Combinatorial Optimization. Technical Report MIT-CSAIL-TR-2009-033, MIT CSAIL, 2009.

[31] J.L. Rogers, S. Aerospace, V. Hampton, J.J. Korte, and V.J. Bilardo Jr. Development of a Genetic Algorithm to Automate Clustering of a Dependency Structure Matrix. Technical report, NASA/TM—2006-214279, February 2006.

[32] G. Rote. The convergence rate of the sandwich algorithm for approximating convex functions. *Computing*, 48(3):337–361, 1992.

[33] S.J. Russell, P. Norvig, J.F. Canny, J. Malik, and D.D. Edwards. *Artificial intelligence: a modern approach.* Prentice hall Englewood Cliffs, NJ, 1995.

[34] J.D. Schaffer. Some experiments in machine learning using vector evaluated genetic algorithms (artificial intelligence, optimization, adaptation, pattern recognition). *PhD Thesis*, 1984.

[35] Theodore K. Seher. Campaign-level science traceability for earth observation system architecting. Master's thesis, MIT, 2009.

[36] Williard L. Simmons. *A Framework for Decision Support in Systems Architecting.* PhD thesis, MIT, 2008.

[37] M.E. Sosa, S.D. Eppinger, and C.M. Rowles. Designing modular and integrative systems. In *ASME Conference on Design Theory and Methodology, Baltimore, MD*, 2000.

[38] N. Srinivas and K. Deb. Muiltiobjective optimization using nondominated sorting in genetic algorithms. *Evolutionary Computation*, 2(3):221–248, 1994.

[39] D. Steward. The design structure matrix: A method for managing the design of complex systems. *IEEE Transactions on Engineering Management*, 28(3):71–74, 1981.

[40] B.S. Stewart and C.C. White III. Multiobjective a*. *Journal of the ACM (JACM)*, 38(4):775–814, 1991.

[41] M. Tamiz, D. Jones, and C. Romero. Goal programming for decision making: An overview of the current state-of-the-art. *European Journal of Operational Research*, 111(3):569–581, 1998.

[42] J. Wallenius, J.S. Dyer, P.C. Fishburn, R.E. Steuer, S. Zionts, and K. Deb. Multiple criteria decision making, multiattribute utility theory: Recent accomplishments and what lies ahead. *Management Science*, 54(7):1336, 2008.

[43] B.C. Williams and R.J. Ragno. Conflict-directed A* and its role in model-based embedded systems. *Discrete Applied Mathematics*, 155(12):1562–1595, 2007.

[44] A. Yassine and D. Braha. Complex concurrent engineering and the design structure matrix method. *Concurrent Engineering*, 11(3):165, 2003.

[45] T.L. Yu, A. Yassine, and D.E. Goldberg. A genetic algorithm for developing modular product architectures. In *Proceedings of the ASME*, pages 2–6, 2003.